Université de Montréal

Compound Terms for Information Retrieval

par
Cuihua Lu

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de
Maître en Informatique

December, 2004

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé:

Compound Terms for Information Retrieval

présenté par:

Cuihua Lu

a été évalué par un jury composé des personnes suivantes:

Esma Aïmeur (président-rapporteur)

Jian-Yun Nie (directeur de recherche)

Philippe Langlais (membre du jury)

Mémoire accepté le 13 janvier 2005

Résumé

Comme de plus en plus d'occasions d'affaire, en particulier, des appels d'offres (ADO) sont disponibles sur Web, il devient important de fournir un système de recherche d'information (RI) efficace pour les découvrir automatiquement. Le projet MBOI (Matching Business Opportunities on the Internet) vise à collecter les informations d'ADO sur les sites Web et à les rendre disponibles aux utilisateurs par des techniques avancées en recherche et en classification. En particulier, un des aspects concerne l'utilisation des termes composés (ou des concepts complexes) extraits par un outil linguistique dans la recherche et la classification. L'étude décrite dans ce mémoire vise à exploiter ces termes dans la recherche.

Dans notre travail, nous avons adopté un système à code ouvert (open source) de RI puissant et flexible - Lucene comme outil d'indexation et de recherche de base. Nous avons tenté d'améliorer la performance de recherche en:

1. reconnaissant des termes composés ou des concepts complexes comme terme d'indexation additionnel;
2. exploitant des stratégies qui utilisent des termes et des concepts composés en combinaison avec des mots clés simples dans la recherche pour augmenter l'efficacité de la recherche ;

Pour ce faire, nous avons implanté le modèle *stream* qui combine plusieurs mécanismes de recherche séparés. Dans notre cas, nous avons créé des *streams* séparés pour des mots clés, des termes/concepts composés et simples. Ces *streams* sont ensuite combinés ensemble.

Nous avons testé notre approche sur des collection de test de TREC – AP et WSJ. La performance globale a été améliorée quand les concepts complexes et simples sont intégrés. En particulier, quand les concepts complexes et les concepts simples sont tous utilisés, nous obtenons des améliorations jusqu'à 8.9%, en comparaison avec la méthode de base qui utilise les mots clés. Ce résultat montre le bénéfice potentiel de concepts complexes et simples pour la RI.

Mots clés : recherche d'information, terme composé, modèle *stream*, expérimentation, TREC

Abstract

As more and more business opportunities, in particular, calls for tenders (CFT) become available on the Web, it becomes important to provide effective information retrieval system to discover them automatically. The MBOI (Matching Business Opportunities on the Internet) Project aims to collect CFT information from web sites and make them available to end users by using advanced search and classification techniques. In particular, one aspect concerns the utilization of compound terms (or complex concepts) extracted by a linguistic tool in search and classification. The study described in this thesis tries to exploit these terms for search.

In our study, we adopted a powerful and flexible open-source IR system - Lucene as the basic indexing and retrieval tool, and try to improve the retrieval effectiveness by:

1. Recognizing specialized compound terms, complex concepts and simple concepts as additional indexes;
2. Exploiting the strategies that using compound terms and concepts in combination with single keywords in order to increase the retrieval effectiveness;

In order to do this, we implemented the stream model which combines several separate retrieval mechanisms together. In our case, we created separate streams for

keywords, compound terms/concepts and simple concepts. These streams are them merged together.

We have tested our approach on the AP and WSJ collections of TREC. The global performance is improved when compound and simple concepts are integrated. In particular, when both complex concepts and simple concepts are used, we can obtain an improvement of up to 8.9% in comparison with traditional IR based on keywords. This result shows the potential benefice of compound and simple concepts for IR.

Keywords: information retrieval, compound terms, steam model, experiments, TREC.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my since appreciation to Professor Jian-Yun Nie, my supervisor, for his invaluable guidance and his great support throughout all the course of this thesis work.

Moreover, I wish to thank Cirano and Nstein technology Inc. for supporting this project. This thesis would not have been possible without their professional support. I would also like to thank Stephane Vaucher, Pascale Bélanger.

Finally, I would like to thank my classmates and friends, Fuman Jin, Zhuo Zhang, Guitao Gao and Qi Zhang with whom I spent a considerable amount of time finding solutions to the problems which arose while pursuing this project.

Chapter 1

Introduction

As the number of publications of business opportunities, in particular, Calls For Tenders (CFT), available online increases, it becomes desirable to provide access to many of this kind of information through a unified information retrieval system. Currently, these Calls For Tenders are found through thousands of listings manually. In order to efficiently discover these business opportunities published on the Internet by automatically scanning, matching and classifying, the MBOI Project (Matching Business Opportunities on the Internet) collects the information from web sites and stores into a database. It then uses advanced search tools, through combing keywords with concepts to search for as possible business opportunities.

The goal of an information retrieval system is to locate relevant documents in response to a user's query. Traditional retrieval system represents documents and queries by the keywords they contain, and rank the documents according to the number of words they have in common. Retrieved documents are typically presented as a ranked list, where the ranking is based on estimations of relevance. Usually the more words the query and document have in common, the higher the document is ranked.

More and more Natural Language Processing (NLP) techniques are applied in IR system, Natural Language Information Retrieval (NLIR) techniques have been experimented on TREC (Text Retrieval Conference) corpora that collections of documents used for testing. The experiments show positive results: robust NLP techniques can help to derive better representation of text documents [Strzalkowski, 1995]. For example, it turns out that phrases or compound terms can be used as search terms in IR. Compound terms are built by combining two (or more) simple words. For example, the compound term "fire wall" has a "combined" meaning as computer device. On information retrieval, it would make no sense to retrieve descriptions containing "fire" or "wall" in response to a query which contains "fire wall". In such a case, the compound term has to be considered as a single term. Phrases or compound terms have been found to be useful indexing units by most TREC performance evaluations of IR [Mitra et al., 1997]. In our study, we use some existing tools such as ExTerm and Concepts Extractor, to identify compound terms and concepts from the raw text of documents and queries, and used them as additional representation of the text for indexing and search purpose rather than simple word methods commonly used. The aim of our study is to see how compound terms should be combined with single words. We tented different approaches with compound terms or concepts to improve performance of retrieval, and these approaches showed substantial improvement in retrieval effectiveness.

The combination of different text representations and search strategies has become a standard technique for improving the effectiveness of information retrieval [Croft 2000]. In particular, the stream architecture where each stream represents an

alternative presentation of the documents and queries was used in NLIR. We will implement stream model to combine keywords and compound terms or concepts in search, and also try to balance their weights by different strategies.

1.1 Problems with the current IR systems

Current information retrieval approaches mostly use keywords or single words stems for search. However, the content of a document or a query cannot be captured precisely by a "bag" of keywords [Arampatzis et al., 1998]. Sometimes the separated words will change the meaning of the compound terms. For example, if the compound term "Hot dog" are represented by the single words "hot" and "dog", its meaning can not be correctly represented. So the words as indexing units may cause different kinds of problem. Polysemy of word is one of the problems. [Arampatzis et al., 2000] mentioned several potential problems with single word indexes:

1. It does not handle cases where one meaning is represented by different words. For example, the result for a query with the keyword *player* does not retrieve documents which contain its synonym *actor*.

2. It does not distinguish cases where single words have multiple meanings, or semantical variation. The word *dog* in the compound term "*Hot dog*" will face with this problem.

3. It does not deal correctly the problem of syntactical variation. For example *science library* is not the same as *library science*.

To solve these problems, in [Zhai et al., 1997] and [Mitra et al., 1997] various retrieval schemas with phrases have be proposed to take into account these problems mentioned above. Phrases or compound terms, have more precise meaning than single words, they are better representation of contents of document and query. Despite the large amount of work in the area the results until now have varied. It is still not clear whether phrases can be used to improve retrieval effectiveness consistently [Pickens, 2000].

In the last several years, concept-based information retrieval tools have been created and used mostly in academic and industrial research environments [Guarino et al., 1999] [Woods, 1997]. Concept-based information retrieval try to search for information based on their meaning rather than on the presence of keywords. A Concepts Extractor extracts concepts with a semantic analysis. The extracted concepts can create a more precise representation of contents of document, and better reflect the user's intention. However, a large number of experiments using words sense do not necessarily lead to improvement of the performance of IR. One of the problem is that, while the extracted compound terms or concepts are meaningful and precise, they usually do not have a complete coverage of the document or query contents. Therefore, using concepts alone, recall will suffer. In our study, we use compound terms and concepts besides keywords, not instead of keywords. These approaches add more precise terms to represent the contents of documents and queries.

1.2 Context and goal of the project

The use of phrases or compound terms as a text representation or indexing units has been investigated since the early days of information retrieval research. Usually one believes that phrases or compound terms have more precise meaning in the document than single word. So phrase indexing assuming that a phrase is a better discriminator than a single word. With phrases or compound terms, one should be able to increase the effectiveness of IR system. Phrases or compound terms can be used as units of indexing. In this case, the relevant documents can be retrieved only if the compound terms occur in them. When using a single word index, a query containing the phrase "information retrieval" will also match with documents containing only "information" or "retrieval" or "information" and "retrieval" separately. If "information retrieval" is recognized as a unit, then these mis-matches may be avoided.

Researchers have used different strategies to identify suitable phrases or compound terms for indexing, namely statistic and syntactic analyses. Statistic analysis aims to identify phrases or compound terms that have a large number of occurrences and co-occurences of its components words in documents. Syntactic analysis aims to identify phrases based on syntactic relations or some syntactic constraints among its component words [Croft, 2000]. The experiments with both types of strategies have proven to be equally successful [Fagan, 1987] and [Salton et al, 1990].

In our work, in order to compare the compound terms and concepts, we used two existing tools for extracting compound terms. Exterm is a simple tool for the

extraction of compound terms from textual documents. It is constructed by a researcher at RALI [Macklovitch, 2001]. Concepts Extractor is a tool of Nstein technologies Inc. It identifies and extracts concepts from texts. Especially, Concepts Extractor also tries to distinguish complex concepts and simple concepts. In addition, it provides a list of candidate concepts that are confirmed to a less degree of confidence.

Our research goal is to define a method to use phrases or compound terms, so that they contribute in increasing IR performance and which can be used effectively.

As basic IR system, we use Lucene search engine because it has some particular characteristics [Lucene 1]. Lucene is an open source search engine. It is easy to modify to satisfy our experiments requirements. Because of the powerful and flexible features of Lucene, various approaches to integrate compound terms or concepts in indexing and searching can be tested. Lucene used vector space model of IR. So we tested the following approaches with compound terms or concepts:

1. Directly add compound terms or concepts to the single words vector;
2. Use compound terms or concepts to replace their compound words;
3. Using keywords, compound terms or concepts representation as different vectors.

The above approaches have been implemented as a stream model. Stream model used several independent, parallel indexes, stream indexes are built using different indexing approaches and weighting strategies, the final results are produced by merging ranked lists of documents obtained from searching all stream indexes. This model is relatively easy to use to combine the different indexing features or

representations. It provides the possibility to merge the results using different techniques.

The MBOI project has a corpus of calls for tenders for experiments. However there are no queries with relevance judgements to evaluate the results. So we used the AP (Associated Press) collection of TREC for our experiments. Once the best strategy is found, we apply it to the call-for-tenders collection of MBOI.

1.3 Organization of this thesis

The thesis is organized as follows. In the chapter 2, we review the main IR concepts, algorithms, and models. The third chapter presents the Lucene search engine system that we will use as an IR system. In chapter 4 and chapter 5 we will describe our approaches and experiments. In chapter 6 we will summarize the main conclusions and outline some possible future research avenues.

Chapter 2

Related work in IR

Natural Language Processing (NLP) techniques have provided useful tools to IR. Phrases (compound words) are important elements of natural language. From the information retrieval (IR) point of view it is also an important factor to increase the retrieval effectiveness. It has been used in both commercial and experimental search engines. Our study will try to develop some new approaches.

We will start from the basic techniques in IR, the document indexing, retrieval models and the stream model. We will also describe some closely related approaches that use phrases and concepts in IR: phrases identification, phrases indexing and their proper weighting.

2.1 Document Indexing

Documents indexing process aims to create a representation of document contents to be used for the retrieval process. The goal of indexing is two-fold. 1) It identifies the most important concepts described in the document; and 2) it measures the importance of each concept in the document [Nie, 2001]. The original documents are usually unstructured texts. Their contents cannot be manipulated directly.

Traditionally in IR systems, keywords are used as the representation units. They are usually single words. Information retrieval systems are largely dependent on the similarity measures according to keywords. Keywords are often weighted in a way that the weight reflects the importance of the keyword in the document.

Usually in indexing process there are three main steps in document indexing process: stop-words elimination, word stemming, and documents term weighting.

1. Stop-words elimination

It is recognized that a word that occurs in 80% of the documents in the collection is useless for purposes of retrieval [Salton & McGill, 1983]. It is clear that not all words in a natural language are meaningful for representing the contents of documents or queries. There are some words that only play linguistic roles without having a semantic meaning, and the frequency of these words is very high. For example in English, some articles, prepositions, conjunctions like the words "the", "that", "for" usually are useless for information retrieval. These words will be harmful to be kept as indexes because they can introduce noise in the retrieval result. A stop-list is set up to contain these words that are considered meaningless. During the indexing process, the system compares each word with the stop-list and removes the stop words.

2. Stemming operation

Stemming is a technique to transform different inflections and derivations of the same word to one common "stem" [Carlberger et al., 2002]. Keywords can appear

in different forms due to morphological variations. Different word forms may have the same meaning. For example, the noun's singular and plural, conjugated verb, etc. such as, "wolf", "wolves", and all of following words "collection", "collections", "collective", "collected", and "collecting". When word suffixes are removed or transformed, similar words will be represented by the same stem "wolf" and "collect". These words will not be considered as completely different during the retrieval process.

Stemming operation can reduce the size of index. For example, the above several words can be transformed in to one stem. Another important benefit is that it can improve the recall of IR. For example, if a user's query is "collection", after remove the "-ion" suffix, the documents retrieved will cover the documents that contain "collected", "collective" and so on. In this way, more relevant documents are found, and the recall can be improved.

To deal with the morphological variants of keywords, a set of stemming rules and some more sophisticated process may be used. Porter proposes a stemming process consisting of several successive steps, each step dealing with one type of suffix. Porter stemmer [Porter, 1980] is one of the most used stemmers for English. This stemming process is integrated in Lucene search engine.

3. Term weighting

Term weighting is an important aspect of text retrieval systems. Terms are words, phrases, or any other indexing units used to identify the contents of a text. Different terms have different importance in a text. Three main components are often

used to determine the importance of a term in a text: the term frequency factor tf, the inverse document frequency factor idf, and document length normalization.

Term weighting plays a very important role for similarity measure. The weight assigned to an index term denotes the importance of the term as an indicator of document or query content. They can be calculated in many different ways. Various *tf* × *idf* weighting scheme are widely used in IR. *tf* indicate term frequency, measuring frequency of occurrence of the term in document or query, and *idf* indicate inverted document frequency, measuring the number of documents contain a query or document term. One of the *tf* × *idf* formulas is as below:

$w_t = [\log(f(t, d)) + 1] \times \log(N/n)$

$f(t, d)$ = the frequency of the term t in the document d,

N = the total number of documents in the collection,

n = the number of documents in which term t occurs

In this formula, $[log(f(t, d)) + 1]$ is the tf factor derived from the term frequency $f(t, d)$, and $log(N/n)$ is what we call *idf*. If n increases idf decreases.

Document length normalization is a way to normalize the term weights for a document in accordance with its length [Singhal et al., 1995]. Cosine normalization is the most commonly used normalization technique in vector space model [Salton et al., 1975]. The cosine factor of normalization is calculated by the following formula:

$$\sqrt{w_1^2 + w_2^2 + \ldots + w_n^2} \qquad (2\text{-}1)$$

where the w_i is the tf×idf weight for a term.

For the purposes of document retrieval, the ideal descriptors are neither very specific nor very general, $tf \times idf$ means that the more frequency a word appears in a document, the more important it is (the tf factor), the less a word appears in different documents, the more specific the word is to the document (the idf factor). So these two factors can be used to weighting a word in documents. The higher the weight of the term is, the more the term is important. These term weights are use to compute a degree of similarity between a query and each document.

2.2 Retrieval models

The representation of the documents and queries for information retrieval tasks has received much attention, and is one of the most important areas in IR. The retrieval model for an information retrieval system specifies how documents and queries are represented, and how these representations are compared to produce relevance estimates [Krovetz, 1992]. The retrieval results are determined according to a retrieval model. In this section we describe the most commonly used traditional retrieval models: vector space model, Boolean model and probabilistic model.

2.2.1 Vector space model

The traditional vector space model attempts to rank documents by the similarity between the query and each document [Salton & McGill, 1983]. The vector space model is the most used model in IR. In vector space model each term corresponds to an independent dimension in a multidimensional space.

The vector space is defined by all the index terms, which are usually all of the words appearing in the document collection during indexing. Each document is represented as a vector of weights. Suppose that there are terms t_1, t_2, ..., t_n is the document collection, then the N dimensional space is defined as follows:

Vector space: $< t_1, \quad t_2, \ ..., \ t_n >$

A document d and a query q are represented as the following weighting vectors, where :

$$d \rightarrow <w_{d_1}, \ w_{d_2}, ..., w_{d_n}>$$

$$q \rightarrow <w_{q_1}, \ w_{q_2}, ..., w_{q_n}>$$

The weights for the document and the query are usually determined according to tf×idf schemas. There are many different ways to measure how similar a document is to a query. The cosine measure is a very common similarity measure. The degree of similarity *sim*(*d, q*) between the query vector *q* and a document vector *d* is calculated using the following formula:

$$Sim(\mathrm{d,q}) = \frac{\sum_{i=1}^{n} (w_{d_i} \times w_{q_i})}{\sqrt{\sum_{i=1}^{n} (w_{d_i})^2 \times \sum_{i=1}^{n} (w_{q_i})^2}} \qquad (2\text{-}1)$$

This similarity measure corresponds [0, 1] to the cosine of the angle formed by d and q vectors. Graphically, this corresponds to the following figure (if we assume that t1, t2 and t3 are all the terms in the vector space):

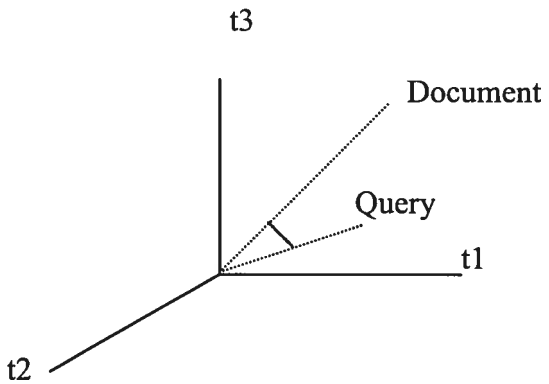

Figure 1 Documents and query represented in vector space

Using a similarity measure, a set of documents can be compared to a query. The documents are then ranked in the reverse order of their similarities to the query.

For example, a document d and query q, we use tf and idf calculate its terms weight, than document d = < 2, 1, 1, 1, 0 > and query q = <0, 0, 0, 1, 0 >, the similarity of the document to the query is:

$$\text{Sim}(d, q) = \frac{2\times 0 + 1\times 0 + 1\times 0 + 1\times 1 + 0\times 0}{\sqrt{2^2 + 1^2 + 1^2 + 1^2 + 0^2} \times \sqrt{0^2 + 0^2 + 0^2 + 1^2 + 0^2}}$$

$$= 1/ (1 \times 2.646) = 0.378$$

In this way, each document's similarity can be measured, then we can obtain a ranked list of documents depend on their similarities.

2.2.2 Boolean model

Boolean model is the oldest of the three [Salton & McGill, 1988] in IR. In this model documents are represented by sets of terms, and queries are represented by Boolean expression with operators. AND, OR, and NOT. In practice, the user just

enters a natural language query, which the system converts into Boolean expressions by using AND or OR to connect different words.

In the classical Boolean expression, no term weighting is used. Boolean system is not able to rank the returned list of documents. To solve this problem, term weighting should be incorporated in a Boolean model. In this way, query evaluation can take into account the weight of the terms. Numerous extension of the Boolean model were suggested to provide a ranked list as result. For example, a fuzzy-set model is suggested by Paice in [Paice, 1984].

The extended Boolean models such as fuzzy-set model support the ranking of relevant documents for boolean retrieval system by using term weights and by calculating query-document similarities. An IR system based on extended Boolean models is defined by the quadruple <T, Q, D, F> [Lee, 1994], where

-T is a set of index terms used to represent queries and documents;

-Q is a set of queries that can be recognized by the system. Each query $q \in Q$ is a legitimate Boolean expression composed of index terms and logical operators AND, OR and NOT;

-D is a set of documents. Each document $d \in D$ is represented by $\{ (t_1, w_{d1}), \ldots. (t_n, w_{dn}) \}$ where w_{di} indicates the weight of term t, in d and w_{di} may take any value between zero and one, $0 \leq w_{di} \leq 1$.

-F is a ranking function

$$F: D \times Q \rightarrow [0, 1]$$

which assigns to each pair (d, q) a number. This number is a measure of similarity between document d and query q, and is called the document value for

document d with respect to query q. Evaluation formulas for the logical operators are the most important factors to determine the quality of document ranking. Following is the evaluation formula based on fuzzy set model, for the evaluation of AND and OR operators:

$$F(d, t_1 \text{ AND } t_2) = \text{MIN}(W_{d1}, W_{d2})$$

$$F(d, t_1 \text{ OR } t_2) = \text{MAX}(W_{d1}, W_{d2})$$

For example, a document *d* and query terms X and Y, suppose the similarity between *d* and X, W_{dX}=0.3, the similarity between *d* and Y, W_{dY}=0.6. The relevant documents for the query (X AND Y), we use fuzzy AND to evaluate as follows:

$$F(X \text{ AND } Y) = \text{MIN}(W_{dX}, W_{dY})$$

$$= \text{MIN}(0.3, 0.6)$$

$$= 0.3$$

Since both terms X and Y are desired, the relevant document is no better than the minimum value.

In practice, the extended Boolean models are also well performing models. Lucene search engine provides Boolean query search, and it implements the fuzzy set model.

2.2.3 Probabilistic model

A probabilistic model attempts to rank documents by their probability of relevance given a query. The documents and queries are represented by binary vectors

$\vec{d}$ and $\vec{q}$, in which each element indicates whether a term occurs in the document or query, or not. The probabilistic model uses odds expression:

$$O(R|\vec{d},\vec{q}) = \frac{P(R|\vec{d},\vec{q})}{P(\overline{R}|\vec{d},\vec{q})}$$

To rank documents, where R means that the document is relevant and $\overline{R}$ means document is not relevant. In order to estimate the probability of the presence or absence of a term among the relevant documents, one uses a set of document samples whose relevance is judged manually. To determine whether a document should be retrieved, the comparison of $P(d_k|R,\vec{q})$ and $P(d_k|\overline{R},\vec{q})$ is often used. Assume that the terms are statistically independent. Then we have [Hiemstra & Vries, 2000]:

$$O(R|\vec{d},\vec{q}) = O(R|\vec{q}) \cdot \prod_{k=1}^{l} \frac{P(d_k|R,\vec{q})}{P(d_k|\overline{R},\vec{q})} \qquad (2\text{-}3)$$

The true probabilities of their relevance to query are not known, probabilistic model estimate the probability with terms' presence and absence. For example, there is document set D, contains terms t_1, t_2, t_3, t_4. In this set, a document d contains t_1, t_3. Then the estimation of probability for d is as follows:

$$P(d|R,\vec{q}) = P(t_1=1|R,\vec{q}) * P(t_2=0|R,\vec{q}) * P(t_3=1|R,\vec{q}) * P(t_4=0|R,\vec{q})$$

$$P(d|\overline{R},\vec{q}) = P(t_1=1|\overline{R},\vec{q}) * P(t_2=0|\overline{R},\vec{q}) * P(t_3=1|\overline{R},\vec{q}) * P(t_4=0|\overline{R},\vec{q})$$

The formula (2-3) can be rewritten into a formula that includes values for term present in the query only, as follow:

$$O(R|\vec{d},\vec{q}) \propto \sum_{\substack{k \in matching \\ terms}} \log \frac{P(d_k \mid R,\vec{q})(1-P(d_k \mid \overline{R},\vec{q}))}{p(d_k \mid \overline{R},\vec{q})(1-P(R,\vec{q}))} \qquad (2\text{-}4)$$

This model assumes that the presence or absence of a term is the only indicator of relevance. It also assumes independence between index terms. This independence assumption is used in formula (2-3).

2.2.4 Discussion

The vector space model and probabilistic model are different approaches to information retrieval. The vector space model is based on the similarity between document and query in a vector space. It also supports partial matching, most of the information retrieval systems are based on vector space model. The probabilistic model is based on the probability of relevance and irrelevance. It considers the distribution of terms over relevant and non-relevant documents. The extended Boolean model combines the term weight to the Boolean expression. In practice, it is difficult to transfer the query into a Boolean expression.

Boolean model allows for complete query expression with logical operators "and", "or" and "not". It has clean formalism and simplicity. However in the unweighted Boolean model, the IR system may retrieve too few or too many documents. The extended Boolean model does not guarantee to assign a high weight

to documents containing both A and B, although the model strongly favours documents which contain both keywords.

Probabilistic model is that risk factor is individualized for each term, and it has been show that some modified probabilistic models are effective for retrieval on large collections. However, the problem of using probabilistic model is that it needs a set of relevant and non-relevant documents to estimate the probabilities of index terms.

The vector space model allows term weighting. Its similarity calculation allows to rank the documents according to their degree of similarity to the query. This model allows flexibility in matching. However, the assumption that index terms are orthogonal is not always reasonable.

Our project uses Lucene search engine system. It is based on the vector space model and cosine similarity measure to retrieval the relevant documents, and provides the ranked list of results. We will present these details in Chapter 3.

2.3 Stream model

The combination of different text representations and search strategies has become a standard technique for improving the effectiveness of information retrieval [Croft, 2000]. In the stream architecture, each stream represents an alternative text index. This architecture was used in the Natural Language Information Retrieval (NLIR). The results show much improved effectiveness of the retrieval.

The stream model uses several independent, parallel indexes that are built for a given collection, each index reflecting a different representation for text documents

and queries. Stream indexes are built using different indexing approaches, term extracting, and weighting strategies [Strzalkowski et al., 1996]. The final results are produced by merging ranked lists of documents obtained from searching all stream indexes with appropriately queries. The final result is often more accurate than any individual streams.

For example, the documents may be represented simultaneously by single word stems, phrases, etc. One can consider each representation as forming a stream: stem stream, and phrase stream. The streams can be organized as follows [Strzalkowski et al., 1998]:

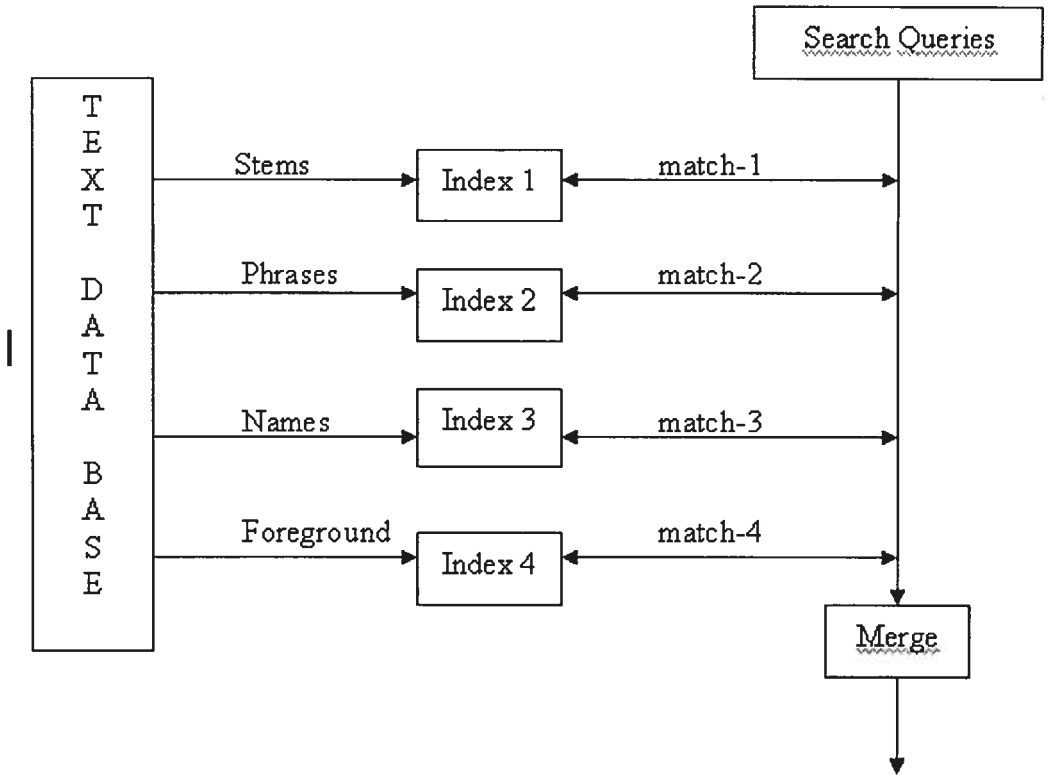

Figure 2 Stream Organization Concept

This model gives the final result from merge results of streams. The combination of retrieval results has received considerable attention in the past few

years, it has been applied not only to merge results from same database as figure 3 shows, but also to merge results obtained from different databases and search engines in a distributed information retrieval environment [Si & Callan, 2002] [Ozaku et al., 2003]. Much research is ongoing on how to merge multiple results from information retrieval systems efficiently and with high quality [Ozaku et al, 03]. In our research, we use multiple IR approaches that might be applied to the same data set. For example, we create different representation to index a set of documents using the Vector Space Model, query the different indexes and get several ranked sets of documents. Then the multiple sets of results will be merged to obtain the final list.

Result merging algorithms are usually compared to a baseline; The goal of a result merging algorithm is to produce a single ranked list that is comparable to what would have been produced from all available databases [Si, 2003]. In general, the Score Normalization method is widely used. This method is adapted for merging information. Normalized document scores are produced from the scores that would have been produced by the database baseline, in such a way that they become more comparable. Based on the normalized score, we merge the streams by balancing the relative importance.

In the experiments in NLIR, some techniques identifying phrases or compound terms as representation of documents like the keyword, this simple approach has proven quite effective for some systems. So we also use the compound terms and concepts as representation of documents and queries to construct a stream model.

In the stream model a more effective stream will carry a higher weight, and larger effect to move the document up in the merged ranking. That means one stream will act as an important role in the final result.

There are several advantages for the stream model. It is relatively easy to combine the contributions of different indexing features or representations. It provides the possibility to merge the results from using different IR engine or techniques. It also can use different sources to increase the performance of a system.

In our experiments, the Lucene search engine provides a convenient index structure, in which each stream can be stored as an independent index field. One can query Lucene with respect to different fields separately to obtain the effect of separate streams. Finally all streams are combined using different ways.

2.4 Using compound term in IR

Usually phrases or compound terms have more precise meaning in the document than single words. Phrase-based indexing assumes that a phrase is a better discriminator than a single word. Using phrases or compound terms in IR system may increase the effectiveness of retrieval logically. Compound terms can be used as units of indexing. Relevant documents can be retrieved in which the compound terms occurs. As this aspect is closely related to our project, we will discuss in more detail about the use of phrases in experimental IR systems.

2.4.1 Phrases as indexing units of documents and queries

Current information retrieval system usually use keywords as units of indexing. Relevant documents are estimated based on a degree of the sharing of keywords between the document and the query. "Retrieval system represent documents and queries by the words they contain, and base the comparison on the number of words they have in common. The more words the query and document have in common, the higher the document is ranked [Salton & Buckley, 1988]." Although keywords have shown to be relatively successful, using words as indexing units may cause different kinds of problem. The most obvious inadequacies originate from linguistic variation, making the Keyword Retrieval Hypothesis insufficient [Arampatzis et al., 2000].

The first is Polysemy of the word, that the single words have multiple meanings due to semantic variation. For example, the documents describing "*acrobat reader*" represent by single words "acrobat" and "reader". The word "acrobat" also have the meaning of "gymnast" or "tumbler". The documents containing the word "acrobat" may not be related to the concept "acrobat reader". If the compound word is separated, the meaning of the compound terms can not be represented correctly by the single words.

The second problem is that it does not deal sufficiently with syntactical variation. For example, *science library* is not the same as *library science*. Although both key words co-occur in the document. The individual words "science" and "library" are not specific enough to distinguish "science library" from "library science".

To solve these problems, phrase representations have been proposed, assuming that a phrase is a better discriminator than a single word. When the retrieval directly uses the compound terms, the documents retrieved will be more accurate. However the recall usually decreases when precision increases. Therefore, when one considers compound terms, one has not to ignore the documents in which only parts of the components of a compound term occur.

In order to improve the quality of index, it is often suggested that phrases be used as descriptors in place of or in addition to general terms [Salton1986, 1983]. Applying the appropriate natural language procedure to extract all instances of compound terms should produce a reasonable representation for documents and queries. Since the indexing term is the representation of a concept, preferably in the form of a noun or noun phrase, an indexing phrase preferably consist of a noun or a noun phrase.

Because the compound terms appear less frequently in the collection than single words, using phrases in place of single words can't lead to good retrieval results. Instead, phrases can be used as units of indexing of the documents and queries together with the single words. In this way, the retrieved documents are those in which the compound term occurs, or its components occur. This combination helps reducing the cases of missing actual relevant documents. Different strategies were used to combine compound terms and single terms in the retrieval process. Phrases have been found to be useful indexing units by most of TREC for performance evaluations of IR systems [Mitra et al., 1997].

Our approaches will use compound terms or concepts as a separate representation of contents, independent of representation of single words. We use NLP technologies to extract compound terms and concepts from documents and queries, these compound terms and concepts are used for indexing both documents and queries. .

2.4.2 Phrase identification

A phrase or a compound term is a sequence of single words. Phrase identification techniques include syntactic, statistical and manual techniques. It is possible to use a thesaurus.

Syntactic phrase identification tries to identify any sequence of words that satisfy certain syntactic relations or constitute specified syntactic structures [Mitra et al., 1997]. The syntactic phrase identification techniques use linguistic rules, both template-based and parser-based, [Salton & McGill, 1983]. In template-based identification, one tags every word in the document with its part of speech (POS), and identifies the syntactic category. Certain tag patterns are recognized by match against a set of syntactic templates. For example, adjacent groups of words from documents are matched against templates, such as <JJ-NN NN> (adjective noun), and <NN PP NN> (non preposition noun), are retained. Most template-based systems are oriented toward finding contiguous words which represent noun phrases. Parser-based strategies attempt to analyze entire sentences or significant parts of them in producing

syntactic phrases [Fagan, 1987]. They try to produce a more complete analysis and then choose some specific linguistic constructs in the pars tree as phrases.

Statistically identified phrases are any pair of non-function words that occur contiguously often enough in a corpus. A statistical phrase identification technique is defined by constraints on the number of occurrences and co-occurrences of its component words and/or the proximity between occurrences of components in a document [Croft et al., 1991]. The basic algorithm tries to select a pairs of words from document and query texts, where the individual words and the form of their co-occurrence satisfy various criteria. Statistic techniques identify phrases by applying the frequency and association statistics to the text. For example, if the words "information" and "retrieval" occur contiguously a large number of times in a corpus, then they would constitute the phrase "information retrieval". This approach produces results that are as good as a syntactic approach.

Thesaurus is another resource for the identification of compound terms. In a thesaurus, a number of compound terms or phrases are stored. Of course the phases in thesaurus are correct terms to be identified. It would be more precise to use those terms as indexes of document contents.

In our study, we used some available tools for compound terms recognition based on linguistic and statistic analysis of documents text.

2.4.3 Appropriate weighting for phrases

Term weight is a crucial part of information retrieval system. Weighting for phrases may differ from weighting for single word terms to allow for the lower frequency and different distribution characteristics [Lewis, 1996]. Most approaches use compound terms as indexing terms in IR, and different weighting strategies are used for compound terms.

An obvious weighting strategy for compound terms is to weight a term as a function of the weights of the components. Since the weight assigned to a descriptor in a vector space model indicates the importance of the descriptor as an indicator of document or query content, the phrase weight may be a function of the phrases elements. So the importance of the elements of the phrase is related to the phrase weight. [Fagan, 1987] assigned a two-word phrase a weight equal to the average weights of its component stems. The stem weights themselves are computed as usual tf×idf.

Given the phrase p in vector V, and the component single words a and b in vector V, the weight of phrase P is:

$$W_{pv} = \frac{W_{av} + W_{bv}}{2} \qquad (2\text{-}5)$$

[Fagan, 1987] presents two reasons for this phrase weighting. First, since the phrase weight is a function of the weight of its components, it incorporates the importance of the phrase elements into the phrases weight. Second, the phrase weight does not differ greatly from the weights of its elements.

Since the frequency of each phrase cannot be higher than that of any phrase components, [Yu & Salton, 1977] assigned the phrase weight as the average weight of the phrase components. This is very similar to the scheme presented above.

Based on this weighting strategy, a document or a query is indexed with both single words and phrases, and this forms two subvectors: one is the single words vector and another is the phrase vector. In order to calculate the similarity between the query vector and document vector, a partial similarity is calculate for each subvector separately, and then the similarity is a weighted sum of two partial similarities. [Mitra et al., 1997] proposed the following formula to calculate the final similarity:

$$Sim_{final} = Sim_{old} + (\textbf{phrase-factor} * \sum_{t \in matchingphrases} w_{doc}(t) * w_{query}(t)) \quad (2\text{-}6)$$

Here the Sim_{old} is the similarity based on matching single words, the second sum is the similarity based on phrases matching. **Phrase-factor** is the relative importance give to phrases matching. It was set manually in the experiments. In [Mitra et al., 1997], the experiments compared the two-word, three-word and longer phrases by using various phrase idf weighting schemes. The results show that there is not much difference between various weighting schemes in vector space model.

Beside the method presented above, many different compound term weighting strategies have been used to improve the effectiveness in IR. [Arampatzis et al., 2000] suggests a simple weighting scheme suitable for phrases which takes into account the

modification structure and its depth. He supposed that phrase frames may contain nested phrase frames at different depths.

In [Chowdhury, 2001] a new adaptive weighting strategy based on query length is proposed. The hypothesis is that "as the query length increases the probability that phrases will degrade effectiveness increases". This means that when the number of phrases increase, highly weighted phrases will cause query drift. For longer query, multiple phrases may over emphasize documents that do not contain all the phrases attributes, and the precision of system decrease. For example, query of "national society legislation department", the phrases "national society", "society legislation" and "legislation department" will over emphasize documents not contain all the terms. So longer query may weighted lower than in short query. The formula for phrase weighting used was:

$$PhWt = \exp(-1*delta*queryLength) \qquad (2\text{-}7)$$

It is supposed that query length is a factor of the phrase weight, and the experiments show that the performance is better than when no phrase weighting is used.

In our project, we will test some new weighting strategies inspired from the above ones.

2.5 System evaluation

IR provides access to vast amounts of information which user desires. However there are so many factors that effect whether or not a user receives the information they needs. The performance of information retrieval system is evaluated

in an objective fashion by verifying whether the answers satisfy the requests as intended by the user.

When we retrieve documents from a large collection, there are four groups of documents: retrieved relevant documents, retrieved irrelevant documents, not retrieved relevant documents and not retrieved irrelevant documents [Evaluate]. A good IR system tries to get maximum retrieval relevant documents and minimum irrelevant documents. We evaluate retrieval by precision and recall.

Precision defines proportion of retrieved document by system that is actually relevant:

Precision = relevant retrieved / total retrieved

Recall defines the proportion of relevant documents actually retrieved in answer to a search request:

Recall = relevant retrieved / relevant in collection

For each query, based on the definitions, we use values of precision and recall pair to make a precision-recall curve as follows:

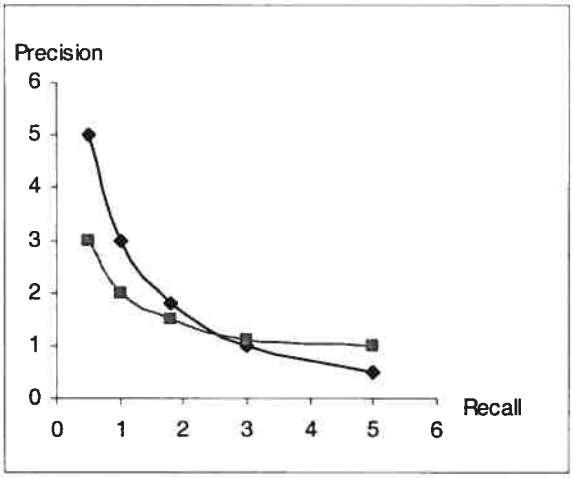

Figure 3 The precision-recall curve for two queries

Figure 2.2 shows that at high precision, recall is usually low, and vise versa. So, there is a tradeoff between precision and recall. IR system typically output a ranked list of documents. So, for each relevant document, one can compute the precision and recall up to a precision, recall curve can be obtained because that point, and average over all precision values computed this way.

In our experiments, we evaluated the results via precisions at 0.1, 0.2,..., 1.0 recall levels. Finally these values at 11 points are averaged together to get an average precision value. The 11-point precisions and the average precision are standard measures used in IR.

Chapter 3

Lucene Search Engine

Lucene is a Java-Based open source search toolkit from Apache's Jakarta project. It provides a Java library that can add text indexing and searching capabilities to an application. Lucene provides powerful APIs. Before using it, one needs to know about Lucene classes and methods.

Lucene uses object-oriented design. It is flexible to be used on any application. In Lucene API, two main processes, text indexing and text searching, are relatively independent of each other, although indexing naturally affects searching. Lucene provides strong functionalities in indexing and searching such as: seachable email, online documentation search, searchable webpages, website search, content search, version control and content management, newwire service feeds [Lucene 1]. It is easy to use. Due to these reasons, in MBOI project we chose Lucene as our search engine.

Lucene was originally written by Doug Cutting, The latest version 1.4 was released July 1st, 2004.

3.1 The Organization of Lucene system

Lucene contains several APIs. The following figure shows an overview of the whole packages:

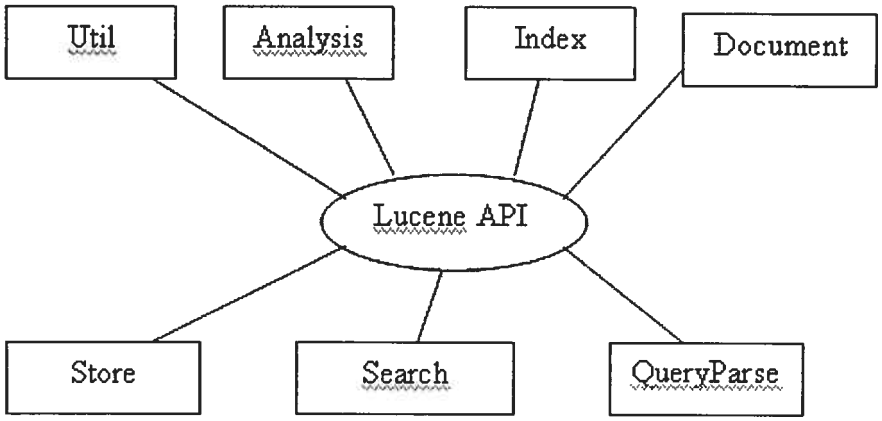

Figure 4 Lucene API

Where:

-Util package contains some data structures, e.g., BitVector and PriorityQueue.

-Analysis package defines some analyzer for converting text into Tokens, and provide implementations such as stop word removing and stemming.

-Document package provides document instance which consists of a set of Fields, Document objects are stored in the index.

-Store package defines classes for storing persistent data, one of them is files directory to store files, and another is RAM directory to implement files as memory resident data structures.

-QueryParser package provides implementation of generating query objects. It parses user query, specifies search field and analyzer.

-Index and Search packages are the heart of Lucene. Index provides implementation of creating the representation of data contents. Search package implements searching on the index to get results out.

3.2 Indexing process

Creating and maintaining index is the first important step for building a search engine. Most of search engines use B-trees data structure to maintain the index; Lucene takes a slightly different approach, rather than maintaining a single index, it builds multiple index segments and merges them periodically. Lucene adds new document to index by first creating a new index segment, then merging it with larger segments [Lucene 2]. So it can maintaining small number of index segments to make search fast. The IndexWriter class has a mergeFactor, it can be used to assign the number of segments in an index.

The core elements of an index for Lucene are segments, documents, fields, and terms. An index consists of several segments, a segment consists of a series of files. Each segment contains one or more documents. Documents are objects. Each

document has one or more Fields, Fields are also objects, and each Field contains one or more terms that indicate a field name and value. A term is the smallest piece of a particular field.

Lucene offers different types of Fields: Text, Keyword, UnIndexed, UnStored. One can choose which type to be used depending on how one wants to use the field and its values. Fields have three attributes of interest: Stored, Indexed, and Tokenized. If a field is Stored, the original text is available to be returned from a search. Indexed makes the field searchable. If a field is Tokenized, field add the text run through an analyzer and token into terms.

The fundamental Lucene classes for indexing text are IndexWriter, Analyzer, Document, and Field. IndexWriter creates a new index and adds Documents to an existing index. Before a text is indexed, it is passed through an Analyzer, which extracts indexable tokens out of the text, eliminates the rest, adds to Fields, and then adds to Documents. Lucene offers several Analyzers for different requirements. Analyzers are components that pre-process an input text. They are also used during searching, because the query has to be processed in the same way the indexed text was processed.

The organization of indexing process is as follows:

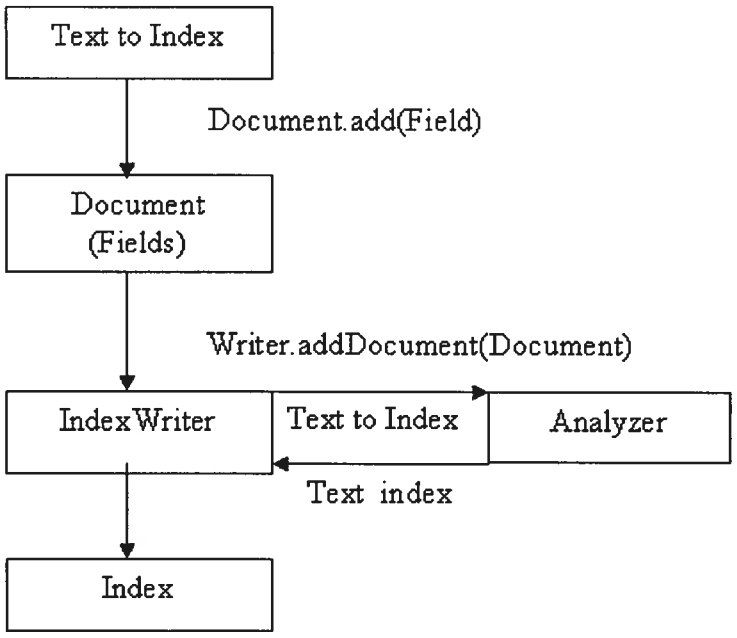

Figure 5 Lucene indexing process

To create an index, first of all, we should define Document object and its Fields, and define how many and what kind of Fields in a Document object, and use add() method in Document class to convert the index files to Document objects.

Then one reads the text to index and convert to Documents object, then uses Analyzer to process the text, and writes the Documents to index by method addDocument() in Writer class. During indexing, the IndexWriter calculates the TF and IDF, and saves them to index. These values will be used to calculate the similarity during searching process.

For example, we want to create an index for articles that contain title, author and text body. The Document in index can be defined as consisting of three Fields, Field name can be "title", "author" and "body". The "title" Field's value can be

defined as Stored, Indexed, and Tokenized. Stored Field can make original title available to be returned from a search. The "author" Field's value can be Stored, Indexed, and UnTokenized. It is not tokenized, but store and indexed. "body" Field's value can be UnStored, Indexed, and Tokenized. UnStored Field can save space of index, but it can be researched. We convert the article to Document object, use Analyzer to remove the stop words, do stemming operation etc, after processing the text. Then it is added to index.

We create an index by Lucene class via these steps [Lucene4]:

1) Create an IndexWriter instance
2) Locate each files to be indexed by walking through the directory
3) For each file be converted to Document object, add the documents to the IndexWriter instance

Lucene indexing process includes token scanning, token parsing, document inverting, frequency counting, postings sorting, files merging and disk writing [Su, 2002]. The tokenization is done through buffered steam. During parsing, we also remove stop words and use Porter stemming. The document object of indexing generally includes the words, a documents ID, and the frequency of the words within the document. The frequency of word is tf. We then calculate the inverted document frequency, the idf. The search index is a list of postings sorted by word.

A Lucene index segment consists of several files:

- A dictionary index containing one entry for each 100 entries in the dictionary
- A dictionary containing one entry for each unique word

Since Lucene can store the index as flat files, the dictionary index contains offsets in the dictionary file, and dictionary holds offsets in the posting file. This approach reduce disk I/O, so retrieval can be fast.

Some search engines only support batch indexing, once an index is created for a set of documents, adding new documents becomes difficult without re-indexing all the documents. Incremental indexing allows easy adding of documents to an existing index. Lucene supports both type of indexing.

3.3 Searching process

Searching is the operation of locating a set of documents that contain desired content match the requirements specified by query [Lucene 2]. Before searching the Query object has to be populated, the Query object can be constructed through the API using the Query subclasses: TermQuery, BooleanQuery, PrefixQuery, WildcardQuery, Rangquery, and a few others. Query object instance is created through QueryParser. User's input query can be parsed to Lucene's API representation of a Query. The query also is treated by specifying Analyzer, such as the SimpleAnalyzer, StandardAnalyzer etc. The Analyzer should be the same as that used for indexing.

The core Class for searching is the IndexSearcher, handle the method search. A Hits collection is returned. The retrieved documents are ranked by Lucene's similarity score.

To instantiate an IndexSearcher with an argument that tell Lucene where to find an existing index, then search over indexReader by the method search(Query) in class IndexSearcher, it returns a "Hits" object. User can through the method provide by Hits class to format the result for output, that is the relevant documents of searching. The Hits collection is itself not an actual collection of the documents that match the query [Lucene 2]. This is done for performance reasons. It has a simple method to call retrieved documents. The Hits object just provides an entry to the relevant documents list. User can get any number of the documents from the list by Hits class.

The organization of searching process is as follows:

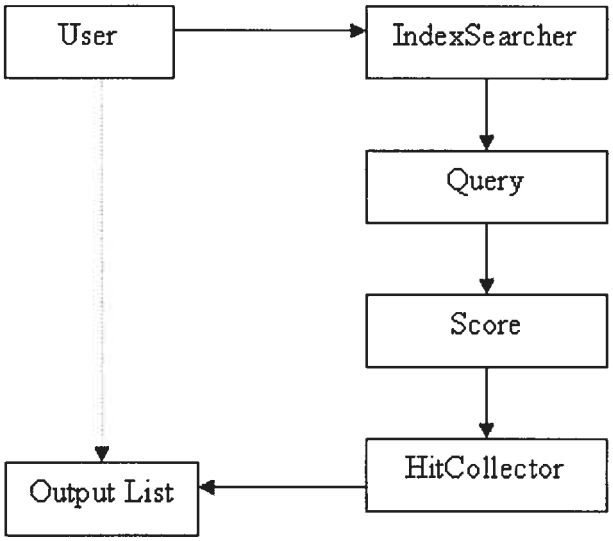

Figure 6 Lucene Searching Process

This searching process includes three phases: issue a search request to index, merge and sort the documents using the score, and take the required "n" documents as output.

In the phase one, the user's queries are converted to Query object by QueryParser class. In other words, Lucene transfers user's query to the query that can be recognized by Lucene. Lucene's QueryParser provides a rich query language, such as, terms search, phrases search, field search etc. IndexSearcher indicates the index to be searched (indexReader), and gives the index name or path to index by a simple parameter.

The phase two implements the Search class and create an instance of hitsCollector, In search process, IndexSearcher implements the score method in Query class, and calculates the score for each document, and ranks documents into hitsCollector.

In the phase three, the user can get the results from Hits object which contains a vector of Document objects. The user can obtain any number of top score documents as output.

3.4 Vector Space Model in Lucene

The documents returned via Hits object are ranked by Lucene's similarity score. The score is the same as the similarity in vector space model. Lucene uses cosine to calculate the similarity [Lucene 5]. The formula is:

$$Score\_d = \frac{\sum_{i=1}^{n} (tf\_q * idf\_t) * (tf\_d * idf\_t)}{\sum_{i=1}^{n} Norm\_q * Norm\_d\_t} * Boost\_t * Coord\_q\_d$$

where:

-tf_q : Square root of the freauency of t in the query

-tf_d: Square root of the frequency of t in d

-idf_t: Log((Number of Documents/$docFreq_t$ +1)+1.0)

-$docFreq_t$: Number of documents containing t

-$Norm_q$: Square root $\sum(tf_q * idf_t)^2$

-$Norm_{d\_t}$: Square root $\sum$(tokens))^2 in d in the same fields as t

-$Boost_t$: The user specified boost for term t

-$Coord_{q\_d}$: The number of terms in both query and document/ number of terms in query.

Lucene's searching process weighting the terms by TF×IDF. $Norm_{d\_t}$ computes the normalization value for a field, and given the total number of terms contained in a field. These values are stored in an index, and in searching process to be used calculate the score. $Norm_q$ is the normalization value for a query and given the sum of the squared weights of each of the query terms. This value is multiplied into the weight of each query term.

Lucene has a saturation for term density. The $Coord_{q\_d}$ is a factor denoting the overlap between a query and a document, for example the number of matching query terms present in a document. The term density saturation functions uses square

root. This "distance werghting" through the $Coord_{q\_d}$ provides a close approximation.

Lucene provides an optional boost factor that can be used to increase or decrease the importance of the term in the document or query. The default value is 1.0. When we calculate the score of query for a document, this factor will be multiplied into the similarity for the term. So $Boost_t$ is used to balance the importance of term in document or query. In our case, we will use this boost factor later to assign different importance to different streams (keyword, compound terms, etc.). It can be assigned at indexing process or at searching time.

Lucene is a technology suitable for full-text search. It supports rapid and incremental indexing. The limit of index usually depend on the system. For example the limit for 32-bits operating system is 2GB. Index size roughly is 30% of the size of the text indexed [Lucene 1].

3.5 Conclusion

Lucene uses object oriented design. The documents stored in the index consist of several filed objects, and each field can be specified in search process. We use this feature to create our stream model. We present how to use Lucene create index for our experiments in next chapter.

Lucene search engine is flexible and powerful, it has strong features like:

1. Incremental indexing allows for easy adding of documents to an existing index without reindexing all the documents.
2. It can index not only files or webpages, but also data from database, or ZIP archive etc.
3. Lucene supports content tagging by treating documents as collections of fields, and supports queries that specify which filed to search.
4. Support non-English search, Lucene preprocesses the input stream through the analyzer class provided by the developer, it is possible to perform language-specific filtering.

Lucene also has some liabilities like:

1. Its main liability is its poor documentation.
2. Lucene is not a web crawler type of engine.
3. It is not available pre-built integration yet.

Chapter 4

Our Approach to IR Using Compound Terms

Our goal is to exploit compound terms extracted by two existing tools for IR. We also apply the Natural Language Information Retrieval (NLIR) stream model in our experiments, in order to create independent index with single words, compound words and concepts.

4.1 Compound terms extraction

In the domain-independent information retrieval, it is difficult to index documents and queries by word senses are more difficult. One identifies phrases or related terms using clues derived from linguistic analysis. The full texts of documents are parsed, dependent word pairs are extracted as indexing units [Strzalkowski et al., 1996]. We use these units to create new representations of documents and queries, or construct new queries.

In Chapter 3, we presented the Lucene search engine. In order to preprocess a document, we use linguistic tools ExTerm and Concepts Extractor to extract the compound terms and concepts from the original text. First of all we introduce Exterm and Concepts Extractor.

4.1.1 ExTerm

ExTerm is a compound term extraction program developed by the RALI. It is based on syntactic templates and statistic analysis. The word sequences are extracted by syntactic templates. The syntactic templates are defined manually according to the syntactic structures of natural language. The procedure of compound terms extraction as follow:

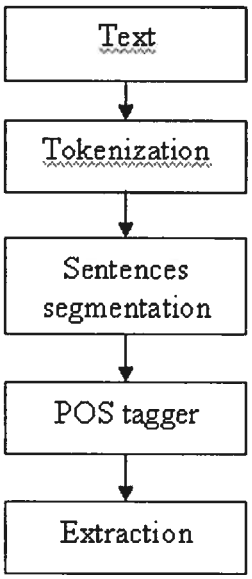

Figure 7 Procedure of compound term extraction

Given a text as input, the program first parse texts into words, sentences and paragraphs, then a POS (Part Of Speech) tagging is used to recognize the syntactic category of each word, such as NN, NC or PP etc. NC means a common noun, AJ an adjective and PP a preposition. It then eliminates all the inflected forms, reducing each word to its base form. Once this is done, ExTerm searches the tag sequences for syntactic templates, syntactic templates are some patterns that correspond to its definitions of multi-word terms in English. For example, two common nouns preceded

by an adjective may form a compound term. Finally, it extracts these candidate terms and sorts them in order of descending frequency; any multi-word sequence that occur more than a threshold in the text is proposed as potential compound terms. An example of text in AP and ExTerm output is as follows:

```
<TEXT>
The state-run Bank of China has joined in a syndicated $50
million loan to a Soviet bank, marking the first time China has
provided such credit to the Soviet Union, the official China
Daily said Tuesday.
The daily said the nation's special foreign exchange bank is
participating in a loan repayable over seven years to the Soviet
bank Vnesheconombank.
Other participants include Postipankki and Skopbank of Finland,
Algemene Bank of the Netherlands and Denmark's Copenhagen Bank
and Provinsbanken. The report did not say how much the Bank of
China was contributing to the loan package.
The loan is another reflection of growing economic contacts
between China and the Soviet Union that have accompanied a
recent warming of political relations between the two former
Communist adversaries.
The report said China also has joined in syndicated loans to
Yugoslavia and Hungary. It said the Bank of China enjoys a good
reputation in the West, and banks in the Soviet Union and
Eastern European countries have shown a strong interest in
dealing with it.
</TEXT>
```

Figure 8 An example of Text in collection AP

```
3     Soviet Union
2     Bank of China
1     strong interest
1     special foreign exchange bank
1     political relation
1     loan package
1     good reputation
1     former Communist adversary
1     economic contact between China
1     Eastern European country
```

Figure 9 Output of ExTerm

The syntactic templates are used in ExTerm as:

((NC | AJ)) * ((NC | AJ) | NC PP) (NC | AJ)) * NC

The statistic analysis is based on the relative frequency of the word sequence in a text. A certain threshold was used, if the frequency of a sequence occur in the text more than the threshold, the words sequence is consider as a compound term. The higher the threshold is, the less compound terms extract are, however the more the terms are precise. The lower threshold can obtain more compound terms. In our experiments, the threshold is set at lowest point.

4.1.2 Concepts Extractor

Concepts Extractor is a tool of Nstein technologies Inc.. It deals with the identification and extraction of concepts to textual documents. This Concepts extraction system does not use any predefined dictionaries or thesaurus. It identifies and extracts concepts using a sophisticated combination of linguistic-based and statistics-based processes. The algorithm of extraction is based on Nstein's extensive research in computational linguistic, it has resulted in a powerful text-analysis technology, that combines the strengths of linguistics, statistical analysis and artificial intelligence to produce search results with high precision and relevance.

Concepts Extractor extracts complex concepts and simple concepts. The complex concepts extraction is based on M.Guilber's concept of terminological patterns presented in his article [Nstein, 2003]. There are several modules as following:

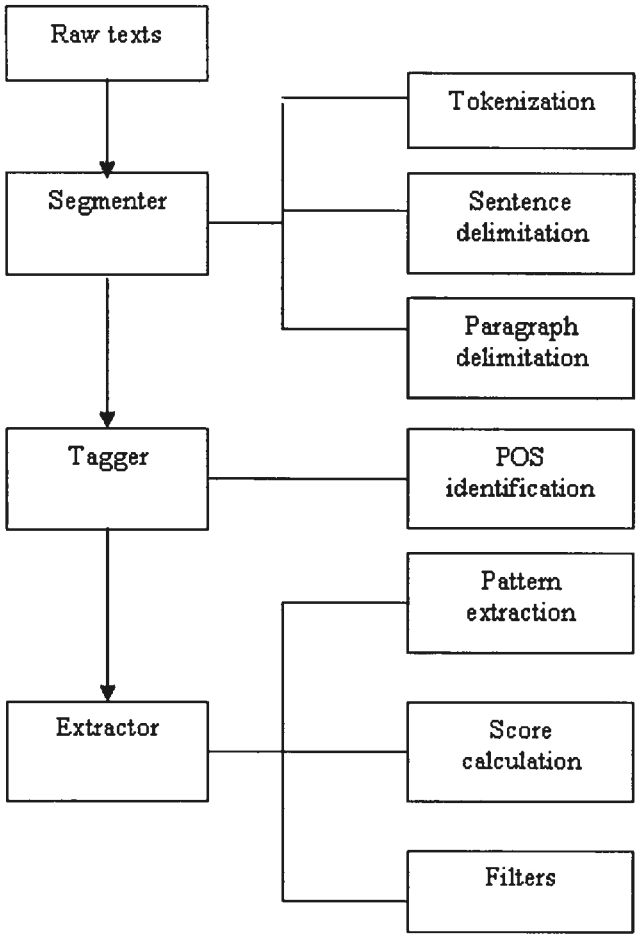

Figure 10 Modules of concepts extraction

The Segmenter module parse texts into linguistic units: generally, words, sentences, or paragraphs. The Tagger is a module which determines the part of speech of each word of the text such as, noun, verb, adjective, etc. There are three steps in the Tagger process: searching in dictionaries, disambiguation and refinement [Nstein, 2003]. The Segmenter and Tagger modules are very similar to the ExTerm.

The extractor module extracts two types of concepts: complex concepts and simple concepts. Complex concepts are extracted by lexical patterns from the text that have been treated by the POS Tagger. The complex concepts are different from

compound terms in that its meaning is precise and often without ambiguity. It is highly related to a specific domain. The simple concepts were defined as a word that is potentially the holder of a sense. Simple concepts have a specific semantic value relative to the subject treated by the text in which is exists, and can be qualified as concepts are: nouns, adjectives, and verbs.

The extraction of complex concepts and simple concepts also uses thresholds. Some units must undergo more rigorous restrictions, some will be grouped intentionally using different forms of filter, and others will be treated more severely using different grammatical filters. The output concepts are sorted in descending order of relevancy.

The outputs of Concepts Extractor are different depend on commands. We choose the full output concepts with frequency and relevancy. An example of Concepts Extractor output is as follows:

```
 <ComplexConcepts>
oncept Frequency="2" Relevancy="96">prime minister</Concept>
oncept Frequency="1" Relevancy="86">presidential election</Concept>
/ComplexConcepts>
SimpleConcepts>
oncept Frequency="4" Relevancy="100">lange</Concept>
oncept Frequency="2" Relevancy="94">hostages</Concept>
oncept Frequency="3" Relevancy="88">mafart</Concept>
 </SimpleConcepts>
 <Candidates>
andidate Frequency="5" Relevancy="98">Ms. Prieur</Candidate>
andidate Frequency="3" Relevancy="97">Rainbow Warrior</Candidate>
Candidates>
```

Figure 11 Output of Concepts Extractor

The frequency score of concepts is calculated by score calculation module. There are three levels of patterns in Pattern extraction module: level A, level B and level C. Level A is applied to very reliable patterns; Level B is applied to generally reliable patterns and level C is applied to not very reliable patterns. The calculation of the frequency combines the real frequency and the level of pattern.

The relevancy score is determined according to the following criteria:

(1) The concept's frequency in the document;

(2) The level of the pattern that performed the extraction;

(3) The pattern's internal constituents;

For example, a "common_noun" tag at the beginning of a pattern is more reliable than an "adjective" tag. A "common_noun" + "common_noun" pattern can extract "office manager", "coffee machine", ... (which are relevant concepts), while an "adjective" + "common_noun" pattern can extract "small manager" , "green table" (which are irrelevant concepts) as well as "dark chocolate" or "yellow fever", which are relevant concepts).

(4) The concept's internal constituents. It use the same way with the pattern internal constituent criteria, but with word lists and/or affixes.

The Concepts Extractor locates and isolates concepts and meanings embedded in documents. It also identifies concepts not defined in categories without the aid of a pre-set thesaurus and uncovers significant connections. So it can identify common and rare concepts in short and long documents.

4.1.3 Collection preprocessing

First, we use ExTerm to extract compound terms from each document, and then adding them to the original document. The process of extraction and adding as follows:

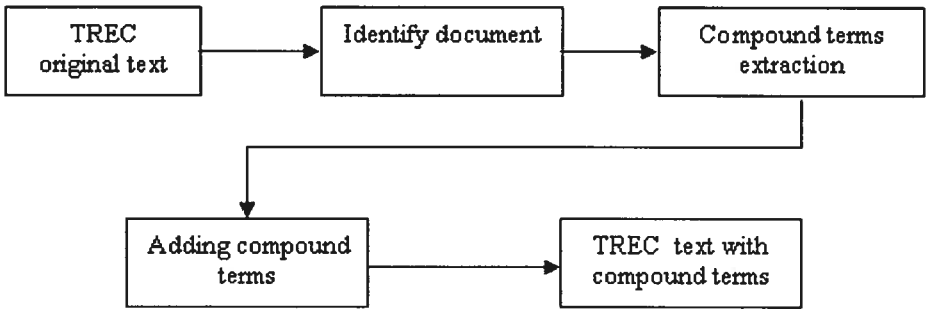

Figure 12 Text preprocessing to extract compound terms

We then used concepts produced by Concepts Extractor of Nstein technologies to form another new corpus.

In order to use the Concepts Extractor, a Java Connector has to be installed, and used to establish a connection with a distant server, called Nserver. The Java Connector is a pure Java program compatible with Sun's JDK 1.2.2 or higher and can be used on any supported platform from the JDK. Any system capable of managing regular TCP/IP connections and standard XML 1.0 data streams can interact with Nserver.

The Java Connector performs the following functions:

- Establishing the connection with a remote Nserver host
- Sending a compliant Nserver XML Command

- Receiving Nserver XML results
- Closing connection to Nserver

The Concepts Extractor can process any text, but the text has to be transmitted to Nserver in XML. Via Java connector, we send the text to Nserver, the returned concepts are added into the original text. This created a preprocessed document collection on which multiple experiments will be possible. The preprocessing steps are shown in the following figure:

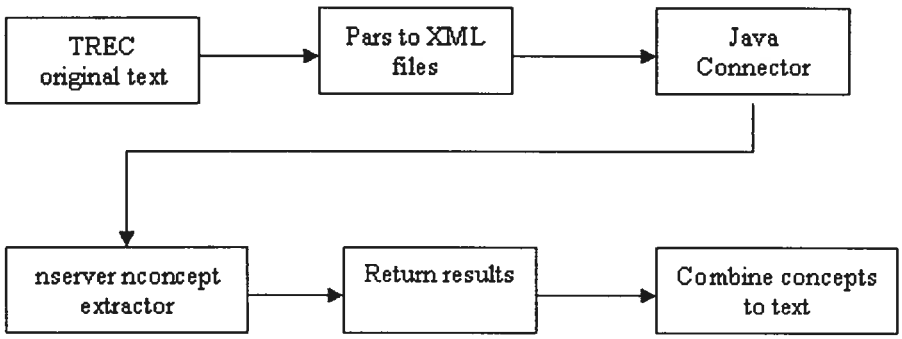

Figure 13 Text preprocessing to extract concepts

4.2 Documents Indexing

We then use Lucene to index the documents. In our indexing we make use of a stoplist (stopFiter class), the Porter Stemma (PoterStemFilter class). The whole indexing process is organized as follows:

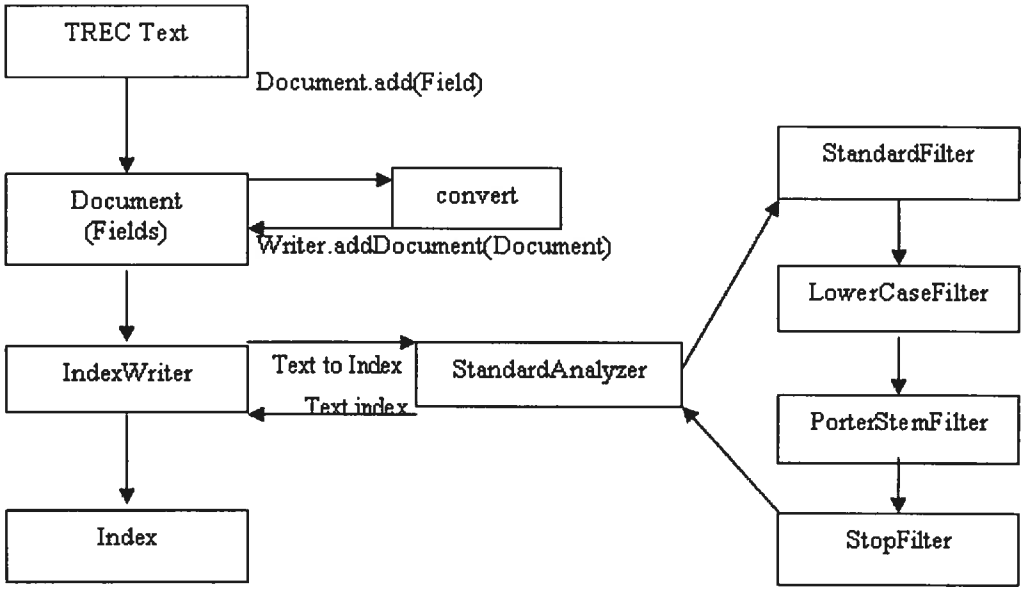

Figure 14 Structure of Indexing

We defined several Fields for a Document object, such as "document number", "contents" etc.

The "contents" Field correspond to the original text, we will use this Field to do keywords searching;

When we extract compound terms with ExTerm, the Document object has three Fields: the document number, the contents, and the compound terms. With Concepts Extractor, a Document object includes five Fields: document number, contents of document, complex concepts of the contents, simple concepts of the contents, and the candidate concepts of the contents.

Each of the above fields is submitted to Lucene for indexing. We can notice that compound terms have been transformed into a form such as computer_science, i.e

the words have been linked with '_', so that Lucene will not cut a compound term into pieces.

4.3 Processing scheme

Lucene search engine is based on vector space model. We used compound terms or concepts as units of index to represent the content of documents and queries. However, compound terms or concepts only cover part of contents of document or query [Nie, 2002]. Therefore, we have to use them in combination with single keywords. We define here several ways to combine them.

First we try to directly add compound terms or concepts to the single words vector. In indexing, the compound terms are put in the same field with the single words. We treat the query in the same way.

Second we attempt to replace the elements of a compound term by the compound term itself. This means that we remove the elements of compound terms from the query, and construct a new query with the compound terms. The index vector again mixes single words and compound terms.

In the last approach, instead of adding the compound terms to single word vector, single words and compound terms or concepts are regarded as different vector space. So a document or a query is represented by two vectors: one for single words and another for compound terms. The calculation of the frequency of the compound terms or concepts is the same as for keywords. That is, the compound terms and concepts are also weighted by the tf×idf weighting scheme. Then we combine these

vectors in many ways, with different importance of compound terms or concepts based on the formula presented in Chapter 2.

This last method is an implementation of the stream model we described in Chapter 2. The streams we use are shown in the following figure:

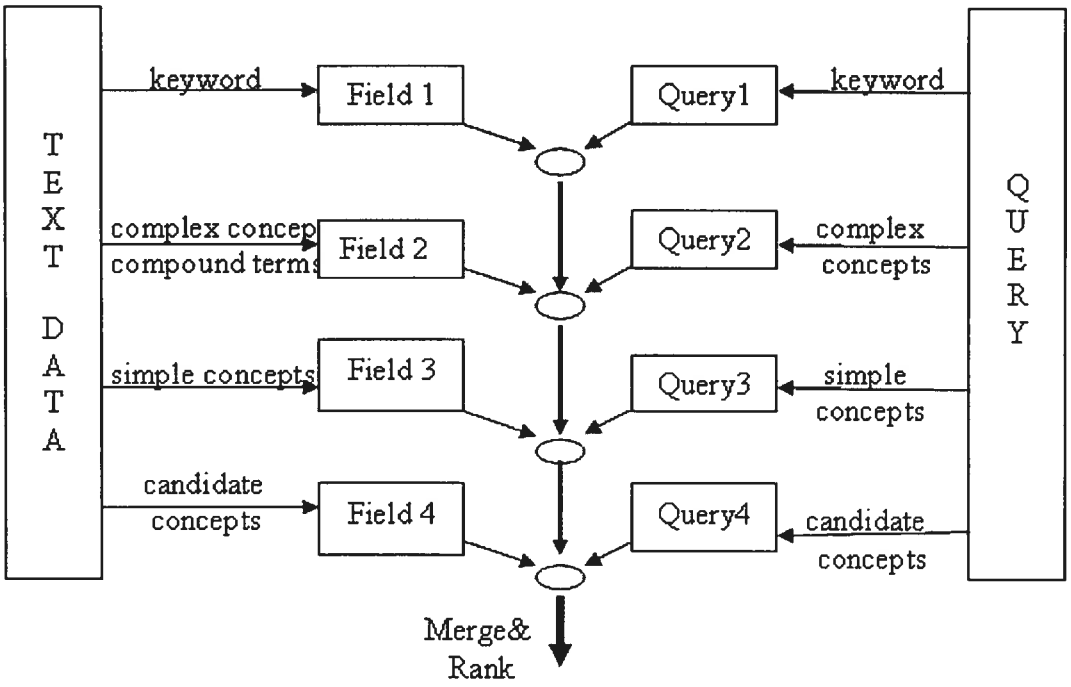

Figure 15 Stream Model organization

4.4 Merging the streams

In the stream model, each stream is evaluated separately to result a similarity measure, then we have to merge the measures to produce a final similarity value. Merging can also be viewed as taking the ranked list of documents obtained from each stream, to produce a single unified rank list.

4.4.1 Using boost factor in merging

First of all, we merged the streams by manually setting a boost factor. We use the formula (2-6) to calculate the similarity between the single terms and phrases described in Chapter 2. We want to use keywords search as our baseline, and observe the contributions of compound terms and concepts. So we keep the boost factor of keywords stream at 1, and assign different boost factors for other streams. Our basic formula for using boost factor is as follows:

$$Sim_{final} = Sim\_{keywords} + \text{Boost-factor} * Sim\_{others} \qquad (4\text{-}1)$$

This formula can balance the importance of each stream by setting a boost factor appropriately. For example, a document compares to a query the similarity is 0.5781 in keywords stream, the similarity in compound terms stream search is 0.4625. In order to keep the keywords search a higher weight, the similarity from compound terms stream search is multiplied by a boost-factor is set between 0-1, such as 0.5. We merge these two streams with adding two similarities, the final similarity for the document is 0.8094. The ranked list of documents obtained from the final similarity.

4.4.1.1 Compound terms or Complex concepts

We merge the fist two streams, i.e keywords and compound terms, by using the formula (4-1), in which keywords stream with compound terms stream or complex

concepts stream. Since the components of the compound term or complex concepts have been considered in the keywords stream, the weight of compound terms or concepts will emphasize the weight of their components. Since the compound terms or complex concepts appear much rarely in the collection, the recall of the retrieval by compound terms alone will be very low. So in our merge process we assign the keywords stream a higher weight.

4.4.1.2 Simple Concepts

The simple concepts are single words, they are also considered in the keywords stream, since they are the important keywords in the documents, we use the Boost-factor2 to emphasize them, in order to increase the weight of these words. Similarly to the same strategy for complex concepts, the Boost-factor2 for the simple concepts can also vary in 0−1.

Simple concepts can be used to match only simple concepts index field (TF). However, since they are also single words, we try another approach that allow simple concepts to math keywords index field (CF).

4.4.1.3 Candidate Concepts

Candidate concepts also are compound terms. They may be names of persons or names of places. We are the same approaches for the candidate concepts as for complex concepts.

4.4.2 Combining boost factor with query length

This strategy is different from the prior work, that phrases weighting is also related to the query length. In Chapter 2, an adaptive weighting strategy based on query length was defined to weight phrases. The idea is when the query length increases, the coverage of the query increases too. The number of compound terms of longer query is bigger than short query, and multiple compound terms may over emphasize documents that do not contain all the terms, this query drift may cause the performance of system decrease. An example query of "natural language information retrieval", the compound terms "natural language", "language information" and "information retrieval" will over emphasize documents not containing all the terms, this cause documents desired can not to be receive a higher ranking. So the compound terms of longer query may be weighted lower than in short query. The wide coverage of query also causes drift of retrieval. So query length is an important factor in IR.

We use the adaptive phrase weighting as boost factor to change the importance of compound terms or concepts. Then the boost factor can be calculated based on query length adaptively. We directly modify the weight of similarity. Depend on the formula of similarity in vector space model, we use the square root of terms weight as phrase weight. We define queryL is querylength and change the formula (2-7) to:

$$\text{PhWt} = \sqrt{\exp(-1 * delta * queryL)} \qquad (4\text{-}2)$$

Where: PhWt is phrase weight.

This phrase weight as boost factor is used for the compound terms to modify the weight of compound terms stream in the final ranking.

In our experiments, we test a set of queries. In order to consider individual query's contribution, we use average query length as a parameter to balance the importance of each query. So we also try to use the query length divided by average query length as another factor, the formula for phrase weight is modified as:

$$\text{PhWtL} = \text{queryL/aveQL} * \sqrt{\exp(-1 * delta * queryL)} \quad (4\text{-}3)$$

Where: PhWtL is phrase weight with average query length.

Here queryL is the query length, aveQL is the average query length in our queries. PhWtL was used as boost factor to modify the importance of the complex concepts.

4.5 Summary

In this chapter, we described the ways that concepts or compound terms are considered in query evaluation. Each type of index is attributed a relative importance, called boost factor. In the next chapter, we will carry out a series of experiments to see whether the incorporation of compound terms can improve retrieval effectiveness, and if it does, what would be the best way to consider them.

Chapter 5

Experiments and Results

The test collection is the AP collection used in TREC. This collection contains 242 782 documents in English from Associated Press. 26 queries are provided with standard answers. These documents are collected by laboratory Bell AT&T in 1988 to 1990. The documents are presented in the standard SGML.

In Chapter 4, we presented two linguistic tools and collection preprocessing procedure. We extract compound terms and concepts from AP collection, preprocess this collection. Using Lucene create independent indexes with keywords and compound terms or concepts. Then we implement several approaches with weighting strategies as a stream model, to investigate linguistic tools ExTerm and Concepts Extractor whether can be used to improve the retrieval system, and the strategies to increase the effectiveness.

5.1 Description of the AP Corpus

Each article in AP collection starts with the label <DOC> and end of the label </DOC>. It includes the document number, the title, and the contents. The contents of

the article is between the marker<TEXT> and </TEXT>. An example of document in AP collection is as follows.

```
<DOC>
<DOCNO> AP880212-0006 </DOCNO>
<FILEID>AP-NR-02-12-88 1644EST</FILEID>
<FIRST>r i AM-CagedHens     02-12 0159</FIRST>
<SECOND>AM-Caged Hens,0162</SECOND>
<HEAD>Court Rules Caging Hens Is Not Cruelty</HEAD>
<DATELINE>STROEMMEN, Norway (AP) </DATELINE>
<TEXT>
   A court ruled Friday that an egg
producer who kept his 2,000 hens in small cages was not guilty of
cruelty to animals, as alleged by animal rights activists.
   ``The verdict is a great relief. It would have been too much to
be found guilty of cruelty to my 2,000 hens,'' Karl Wettre was quoted
as saying by the national NTB news agency after his acquittal.
   The National Society for the Prevention of Cruelty to Animals
claimed that by keeping hens in small cages, Wettre violated
national legislation to allow animals' natural development and
behavior.
   But the court found that Wettre observed Norwegian regulations
stipulating that a hen should have at least 112 square inches of cage
space in which to live.
   NSPCA chairman Toralf Metveit was quoted as saying: ``I'm
disappointed but not surprised.''
   The society was ordered pay $15,600 in court costs.
</TEXT>
</DOC>
```

Figure 16 An Example of document in collection AP

The query is identified by a query number the marker is <num>, each query is separated by <top>and </top>.

```
<top>
<num> Number: 54
<E-title> Dwindling Fish Supplies
<E-desc> Description:
Find documents that discuss the dwindling supplies of fish available
to the commercial fisheries of the European Community.
<E-narr> Narrative:
A relevant document addresses overfishing, disputes concerning fishing
rights, pollution, and other matters which affect the supply of fish.
International agreements aimed at regulating the taking of fish are
relevant.
</top>
```

Figure 17 An Example of Query in AP

In our experiments, we use both the description and the narrative of queries.

5.1.1 ExTerm

We use ExTerm linguistic processing of AP collection, both documents and queries, to extract the compound terms from contents of each article and query. The frequency of the terms in document is also identified. The space in the identified compound term is then replaced by underscore "_".

Then we add a new field into the original document with the label <terms> </terms> after the contents. This field contains the compound terms and their frequency. The number before the compound term is the frequency of the term in this document. The compound terms and it's frequency create new representation of content of document. An example of compound term field of document as follows:

```
</TEXT>
<terms>
      1 square_inch_of_cage_space
      1 natural_development
      1 national_legislation
      1 national_NTB_news_agency
      1 hen_in_small_cage
      1 great_relief
      1 court_cost
      1 animal_right_activist
      1 Prevention_of_Cruelty
      1 Norwegian_regulation
      1 National_Society
      1 Cruelty_to_Animals
</terms>
</DOC>
```

Figure 18 An example of compound terms field

An example of compound terms field of query is as follows:

```
<terms>
      2     supply_of_fish
      1     relevant_document
      1     commercial_fishery
      1     International_agreement
      1     European_Community
</terms>
</top>
```

Figure 19 An example of compound terms field of query

5.1.2 Concepts Extractor

Nstein's Concepts Extractor extracts the following elements: **complex concepts**, **simple concepts**, and **candidate concepts**. The result of document <AP880212-0006> is as follows:

```
<nconceptextractor>
<ComplexConcepts>
<Concept Frequency="1" Relevancy="85">natural development</Concept>
<Concept Frequency="1" Relevancy="62">news agency</Concept>
</ComplexConcepts>
<SimpleConcepts>
<Concept Frequency="3" Relevancy="100">cruelty</Concept>
<Concept Frequency="2" Relevancy="91">society</Concept>
<Concept Frequency="3" Relevancy="83">animal</Concept>
<Concept Frequency="3" Relevancy="76">national</Concept>
<Concept Frequency="3" Relevancy="69">cage</Concept>
<Concept Frequency="3" Relevancy="63">court</Concept>
<Concept Frequency="4" Relevancy="57">hen</Concept>
<Concept Frequency="2" Relevancy="51">wettre</Concept>
</SimpleConcepts>
<Candidates>
<Candidate Frequency="1" Relevancy="97">NSPCA chairman Toralf
Metveit</Candidate>
<Candidate Frequency="1" Relevancy="95">national NTB news agency</Candidate>
<Candidate Frequency="1" Relevancy="91">animal rights activists</Candidate>
<Candidate Frequency="1" Relevancy="89">Norwegian regulations</Candidate>
<Candidate Frequency="1" Relevancy="87">national legislation</Candidate>
</Candidates>
</nconceptextractor>
```

Figure 20 The Result of Concepts Extractor

The three different fields are transformed into different XML segment as follows:

```
</TEXT>
<terms1>
      1 natural_development
      1 news_agency
</terms1>
<terms>
      3 cruelty
      2 society
      3 animal
      3 national
      3 cage
      3 court
      4 hen
      3 wettre
</terms>
<terms3>
      1 NSPCA_chairman_Toralf_Metveit
      1 national_NTB_news_agency
      1 animal_rights_activists
      1 Norwegian_regulations
      1 national_legislation
</terms3>
</DOC>
```

Figure 21 Three terms fields of Document

Once these preprocessings are applied on the AP collection and queries, Lucene can be used to index them.

5.2 Query evaluation using Lucene

Lucene was used as the indexing and retrieval engine. Since the output results of Lucene are a list ranked by score in Lucene system, we get the top 1000 as output. This list is compared with the standard answer, to produce recall-precision values.

First of all, we did traditional keywords search based on Lucene search engine on AP collection, and below is the evaluation result of the keywords retrieval.

Table 1. Evaluation with Lucene search on AP

Recall	Precision(%)
0,00	0,8137
0,10	0,5679
0,20	0,4767
0,30	0,4079
0,40	0,3242
0,50	0,2747
0,60	0,2515
0,70	0,1524
0,80	0,0857
0,90	0,0540
1,00	0,0221
Ave. Precision	29,44

This retrieval results in the average precision of 29.44% for 26 queries.

5.3 Integrating Compound terms and Concepts with boost factor

We compare different boost factor values for two extraction tools, ExTerm and Concepts Extractor.

5.3.1 ExTerm

Recall that the formula used is (4-1) that we repeat here:

$$Sim_{final} = Sim_{\_keyword} + \text{Boost-factor} * Sim_{\_compoundterm} \qquad (5\text{-}1)$$

We tested a series of values for the boost factor of the compound terms in the range of 0.1-1. The results are as follow:

Table 2. Comparison of Boost factor for compound terms with ExTerm

Boost-factor	Average Precision(%)
0	29,44
0,1	29,76
0,2	29,99
0,3	**30,13**
0,4	29,77
0,5	28,84
0,6	28,69
0,7	28,52
1	27,69

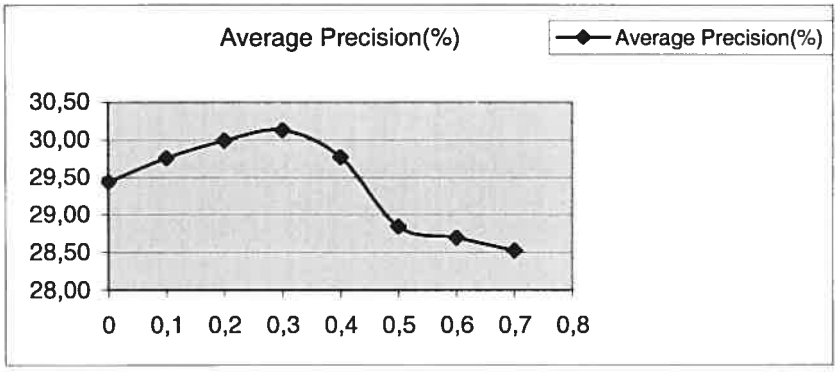

Figure 22 Comparison of Boost factors

When the boost factor is 0.3, the retrieval results is the best. The figure 22 shows that when the boost factor is set between 0.2-0.3, the average precision are slightly better than keyword-based search.

5.3.2 Concepts Extractor

The retrieval streams contain keywords stream, complex concepts stream, simple concepts stream and candidate concepts stream. We attempt to evaluate the impact of the all these concepts on the effectiveness of retrieval.

5.3.2.1 Complex concepts

In the first test, we use complex concepts, in addition of keywords. We tested various value of the boost factor for complex concepts ranging from 0 to 1, while the boost factor of keywords is set at 1.0. The average precision corresponding to different values of boost factor is shown in the as following table:

Table 3. Boost factors for complex concepts from Concepts Extractor

Boost-factor	Average Precision(%)
0	29,44
0,1	30,21
0,2	**30,44**
0,3	29,62
0,4	28,88
0,5	27,37
0,6	26,56
0,7	25,78
1	23,02

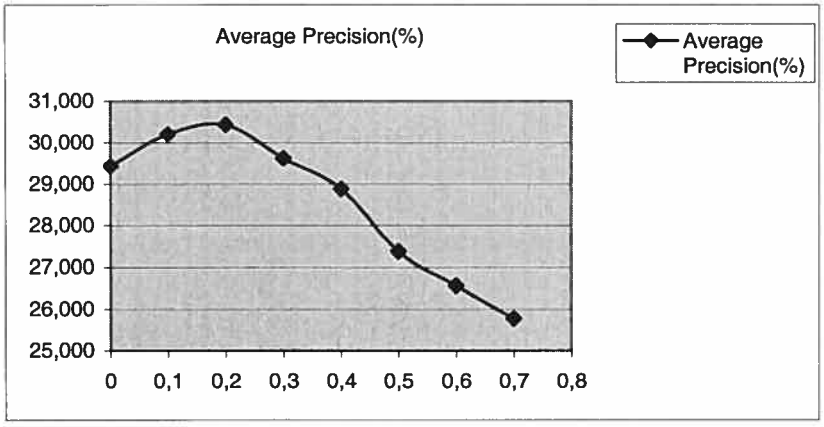

Figure 23 Comparison of Boost factors

The results show when the boost factors were assigned manually between 0.1 - 0.2, the average precision is better than other values and the original searching with keywords only (boost factor=0). When the boost factor is set at 0.2, we obtain the best average precision 30.44%. We notice that we can obtain similar improvements with these complex concepts to those with ExTerm.

5.3.2.2 Simple concepts and candidate concepts

We now use simple concepts to combine with keywords. In this test, the simple concepts of query are only allowed to match simple concepts of a document.

Table 4. Boost factors for simple concepts match simple concepts

Boost factor	Average Precision (%)
0.1	29.97
0.2	28.19
0.3	27.03
0.4	25.87

The results of experiments show that boost factor is 0.1 the result is slightly better than the original searching. The average precision is 29.97%.

Since the simple concepts also are single words, we can allow simple concepts of query to match keywords index of document. This allows us to consider the case where a keyword is not identified as a simple concept in a document while it is identified as a simple concept in a query.

Table 5 Boost factors for simple concepts match keywords

Boost factor	Average Precision (%)
0.1	30.29
0.2	30.31
0.3	30.12
0.4	29.99
0.5	29.19

The results show boost factor between 0.1-0.2 the average precisions are better. Boost factor=0.2, we obtain the best average precision 30.31%. It is much better than matching only with simple concepts of documents.

We use the same approaches on candidate concepts stream. We use candidate concepts to match candidate concepts index Field of documents. We obtain the best average precision 29.42%.

The results of experiments show that the candidate concepts have negative impacts. So we conclude that candidate concepts are not useful for IR.

5.3.2.3 Complex and Simple concepts

We now merge several types of concepts: the keywords stream, complex concepts stream and simple concepts stream. The boost factor of simple concepts is 0.2. The boost factor assigned to the complex concepts stream is 0.2. The final output list obtains that the average precision is 31.33%, an increase of 6.42% with respect of the original searching.

From these results of experiments, the best results of each approach are shown on the table 4. Here TF means the simple concepts match simple concepts index Field. CF means that simple concepts match keywords index Field.

Table 6 Comparison of compound terms with concepts

Manually Boost-factor	Exterm(%)	Concept extractor(%)	Increase(%)
	Average precision		
No compound terms	29,44	29,44	
Compound terms	30,13		(+2,34)
Complex concepts		**30,44**	**(+3,4)**
Simple concepts (TF)		29,97	(+1,8)
Simple concepts (CF)		30,31	(+2,96)
Complex+Simple (CF)		**31,33**	**(+6,42)**

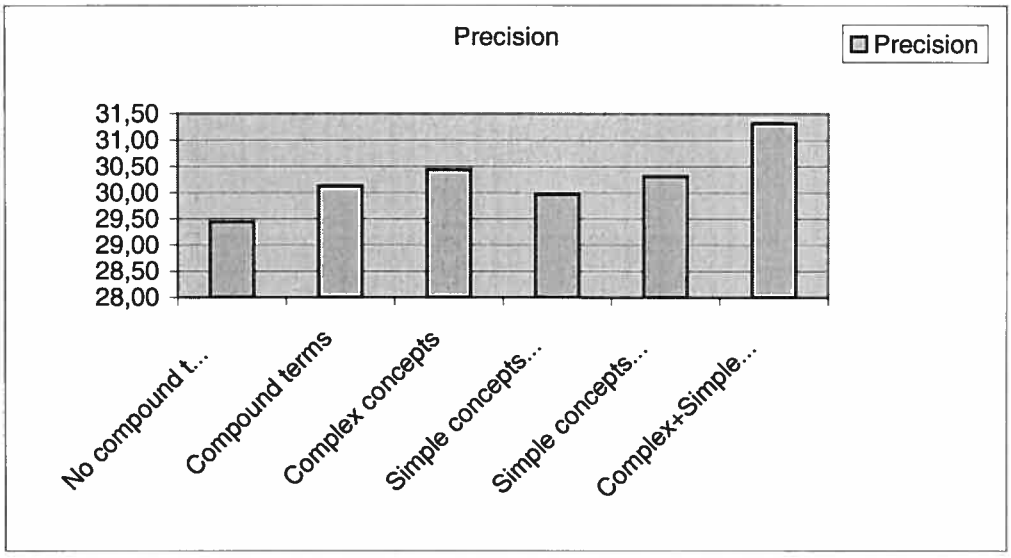

Figure 24 Comparison of compound terms, concepts

We observe that the number of compound terms produced by ExTerm is much more than the number of complex concepts produced by Concepts Extractor. The threshold of Concepts Extractor is much higher. But the complex concepts obtain better retrieval results than compound terms of ExTerm. This means that too many compound terms may create much noise, thus decrease the retrieval effectiveness. The complex concepts and simple concepts from Concepts Extractor with a reasonable assignment of boost factor can lead to relatively large increases in the effectiveness of IR.

5.4 Setting of boost factor according to query length

The boost factor we tested is set uniformly to all the queries. We attempt to determine the boost factor according to query length. Query length is an important

factor when dealing with phrases. We use the adaptive phrase weighting formula (4-2) introduced in Chapter 4, and use phrase weight as boot factor to modify importance of concepts. This boost factor is used to modify similarity directly. We define queryL is query length. The new boost factor is as follows:

$$\text{Boost-factor} = \sqrt{\exp(-1 * delta * queryL)} \quad (5\text{-}2)$$

This formula is used to test complex concepts stream, simple concepts stream and candidate concepts stream one by one.

5.4.1 Complex concepts

The previous experiments showed that the best uniform boost factor is 0.2. The average query length for the set of query in our experiments is around 36. So from the formula (5-2), we calculate that the delta is around 0.1. We test a series of values around 0.1, the results show the delta at 0.1 for this set of queries is the best one. We use this Boost-factor in the experiments for complex concepts. We merged keywords stream and complex concepts stream and get the average precision of 30.17%. This is slightly lower than what we obtained previously (30.44%).

In our experiments, there are a set of queries for AP collection. The results are average for all the queries. In order to consider individual query's contribution, investigate the effect of query length, we use average query length as a parameter to balance the boost factor of each query. So we use the query length divided by average query length as another factor.

For complex concepts searching, we also use formula (4-3) phrase weight with average query length as boost factor. The aveQL is the average query length. The boost factor is as follows:

$$\text{Boost-factor 1} = \text{queryL/aveQL} * \sqrt{\exp(-1 * delta * queryL)} \quad (5\text{-}3)$$

In our experiments, the delta is assigned 0.1, the Boost-factor1 as boost factor of the complex concepts stream. We merge the keywords searching and complex concepts searching, the result obtains retrieval average precision is 30.55%, which is almost the same as previously without query length factor.

5.4.2 Simple concepts

The boost factor for simple concepts is now determined by the following formula:

$$\text{Boost-factor 2} = \sqrt{\exp(-1 * delta * queryL)} \quad (5\text{-}4)$$

We test a series of values of delta around 0.1, and the value 0.12 for delta turns out to be the best. When this new boost factor for simple concepts is used, we obtain an average precision of 31.05%, which is higher than that in Table 4 (30.31%).

5.4.3 Complex concepts and Simple concepts

We combine complex concepts stream and simple concepts stream with their respective boost-factors. The final result shows an average precision of 32.06%, or an increase 8.9% in comparison with the original searching with keywords.

Table 7 Comparison of Automatic Boost

Automatic Boost-factor	Exterm(%)	Concept extractor(%)	Increase(%)
	Average precision		
No compound terms	29,44	29,44	
Compound terms	30,23		(+2,68)
Complex concepts		30,55	(+3,77)
Simple concepts (CF)		31,05	(+5,47)
Complex+Simple (CF)		**32,06**	**(+8,90)**

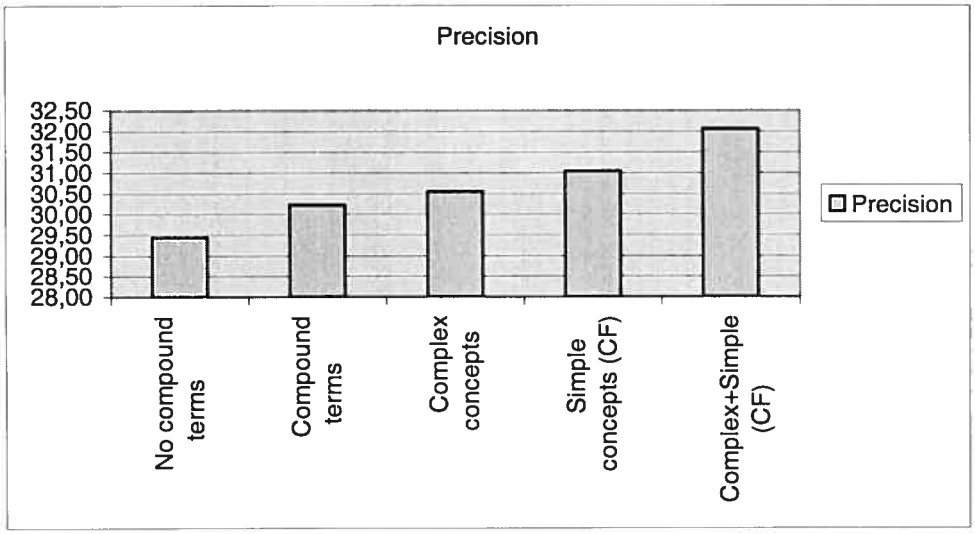

Figure 25 Comparison of Best Results

We observe these results that the query length as a important factor can further improve the effectiveness of retrieval.

5.5 Experiments on WSJ Collection

In order to confirm these results on a different collection, we use the same boost factor on a new test collection – WSJ (Wall Street Journal) collection used in TREC. We use collections WSJ90, WSJ91 and WSJ92 which contains 74,520 documents in English, 50 queries with standard answers [Savoy, 1997].

Wall Street Journal (1990, 1991, 1992) documents are collected from the Federal Register (1988), Associated Press (1988) and Information from the Computer Select disks (Ziff Davis Publishing 1989, 1990).

We preprocess the collection in the same way with AP collection. We also create keywords index, complex concepts index, simple concepts index and candidate concepts index. In our experiments, we extracted concepts from <LP> and <TEXT> sections.

```
<DOCID>
910130-0146.
</DOCID>
......
<LP>
ST. LOUIS -- Anheuser-Busch Cos. once again widened its lead
over other brewers by selling a record 86.5 million barrels of
beer last year.
The company said the 7.2% volume gain over 1989 gives it a 43.7%
share of the U.S. beer market. The previous year, Anheuser held
42% of the market, according to industry estimates.
</LP>
<TEXT>
Anheuser said it now holds a 43 million barrel lead over its
closest competitor, the Miller Brewing Co. unit of Philip Morris
Cos.
Anheuser attributed its sales gain primarily to new brands,
including Bud Dry Draft, launched nationally last
April.
</TEXT>
<Complex>
1 closest_competitor
1 beer_market
1 ST._LOUIS
</Complex>
<Simple>
2 beer, beer
2 market, market
3 anheuser, anheuser, anheuser
2 barrel, barrel
1 industry
</Simple>
<Candidates>
1 Miller_Brewing_Co._unit
1 Anheuser-Busch_Cos.
1 Bud_Dry_Draft
1 Morris_Cos.
</Candidates>
</DOC>
```

Figure 26 An example of WSJ Document and Concepts

5.5.1 Complex concepts

We use complex concepts, in addition of keywords. We tested various value of the boost factor for complex concepts ranging from 0 to 1, while the boost factor of keywords is set at 1. The average precision corresponding to different values of boost factor is shown in the as following table:

Table 8 Boost factor for complex concepts

Boost-factor	Average Precision(%)
0	15,55
0,1	15,60
0,2	**15,88**
0,3	15,77
0,4	15,38
0,5	15,16
0,6	14,99
1	13,43

When the boost factor is set at 0.2, we obtain the best average precision 15.88%. We notice that we can obtain similar boost factor with these complex concepts to those with AP collection, an increase of 2.12% with respect of the original searching.

5.5.2 Simple concepts

We now use simple concepts to combine with keywords. In this test, the simple concepts of query are allowed to match original text of a document.

Table 9 Boost factor for simple concepts

Boost-factor	Average Precision(%)
0	15,55
0,1	**16,14**
0,2	15,66
0,3	15,76
0,4	15,74

When the boost factor is set at 0.1, we obtain the best average precision 16.14%. We notice that we can obtain similar boost factor with AP collection, an increase of 3.79% with respect of the original searching.

5.5.3 Complex and Simple concepts

We merge several types of concepts: the keywords stream, complex concepts stream and simple concepts stream. The boost factor of simple concepts is 0.1. The boost factor assigned to the complex concepts stream is 0.2. The final output list obtains that the average precision is 16.39%, an increase of 5.40% with respect of the original searching.

5.6 Conclusion

We compared the compound terms or concepts extracted with two linguistic tools, ExTerm and Concepts Extractor for IR. The number of complex concepts produced by Concepts Extractor is less than the compound terms produced by ExTerm. This means that Concepts Extractor uses more restrict for criteria extracting concepts. Concepts Extractor not only uses extraction patterns but also uses different filters. The results of all experiments are summarized in the following table:

Table 10 The results of experiments

	Exterm	Concept Extractor	Increase(%)
	Average precision (%)		
Boost factor			
No compound terms	29,44	29,44	
Compound terms	30,13		(+2,34)
Complex concepts		**30,44**	**(+3,4)**
Simple concepts (TF)		29,97	(+1,8)
Simple concepts (CF)		30,31	(+2,96)
Complex+Simple(CF)		**31,33**	**(+6,42)**
Query length			
Compound terms	30,23		(+2,68)
Complex concepts		30,55	(+3,77)
Simple concepts(CF)		31,05	(+5,47)
Complex+Simple(CF)		**32,06**	**(+8,90)**

The above results are obtained on the AP collection. Our experiments show that concepts (both simple and complex) are combined with keywords with reasonable boost factors, the retrieval effectiveness can be greatly improved (up to 8.9% relative increase).

The tests on WSJ collection confirm that using the concepts with boost factor, the IR effectiveness can be improved.

Finally, the approach described in this thesis is also implemented for a collection of Call For Tenders (CFT). When a user issues a query on CFT, we use both simple and complex concepts extracted by Concepts Extractor in contribution with keywords. Unfortunately, for CFT, we do not have test queries with relevance judgments. This is why our experiments have been carried out mainly on TREC collections. We hope that

the settings reasonable for TREC collections would also be reasonable for CFT, although these may be some slightly differences in the optimal values of boost factors.

Chapter 6

Conclusions

In this chapter, we will draw some conclusions from our work.

To solve the problems for professional users who look for Call for tenders (CFT) published on the Internet, we use some linguistic tools to extract concepts from both queries and CFT. The extracted concepts are added as additional representation of contents of the text. Our study described in this thesis is part of information retrieval phase of MBOI project, i.e. using compound terms for retrieval.

In MBOI project, we chose Lucene to be basic search engine in the system. Based on Lucene's flexible features, we focused on two aspects in our study: Indexing with compound terms or concepts, and different search strategies with compound terms and concepts.

- Indexing with compound terms or concepts

Lucene search engine system is based on vector space model of retrieval. Instead of adding the compound terms or concepts directly to single words in the same vector, compound terms or concepts are regarded as different vector space. So a query or a document is represented by several different vectors.

- Searching strategies with compound terms or concepts

In order to enhance retrieval performance, we use compound terms or concepts as a different stream in the stream model. We try to use boost factor of Lucene to set reasonable weights to different streams. Several strategies to determine boost factors have been tested.

On the AP collection, with a simple setting of boost factor for compound terms from ExTerm, we obtained an improvement of 2.34%. We obtained an improvement of 3.4% by integrating complex concepts from Concepts Extractor. If we integrate both simple and complex concepts of Concepts Extractor, we obtained an improvement of 6.42%. With some changes in the boost factor to the query length, we have been able to obtain an improvement of 8.9% on the AP collection.

On the WSJ collection, we have been able to obtain similar results. This shows that the use of concepts in combination with keywords may be an approach that is generally applicable. We hope that this would lead to a similar conclusion on a CFT collection.

Reference

[Arampatzis et al., 1998] Arampatzis, Tsoris, T., Koster, C.H.A., Weide, Th.P., "*Phrase based Information Retrieval*" , Computing Science Institute, University of Nijmegen, the Netherlands, 1998.

[Arampatzis et al., 2000] Arampatzis, A. T., Weide, Th.P., Bommel, P.van, Koster, C.H.A., "*Linguistically-motivated Information Retrieval*" Jan 19, 2000.

[Croft, 2000] Croft, W.Bruce "*Combining Approaches to Information Retrieval"* in Advancees Information Retrieval: Recent Research from the CIIR, Kluwer Academic Publishers, chapter 1, pp. 1-36, 2000.

[Croft et al.,1991] Croft, W.B. Turtle, H.R. and Lewis , D.D. "*The use of phrases and structured queries in information retrieval*" In Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in information Retrieval(Oct. 13-16, Chicago, I11.) ACM-SIGIR, New York, pp. 32-45,1991.

[Carlberger et al., 2002] Johan Carlberger, Hercules Dalianis, Martin Hassel, Ola Knutsson "*Information Retrieval for Swedish using Stemming*" ACM Conference on Research and development in information retrieval, the American Society for Information science, pp. 1-5, 2002.

[Chowdhury, 2001] Abdur Chowdhury "*Adaptive Phrase Weighting*" International Symposium on Information Systems and Engineering (ISE 2001), June 2001.

[Evaluate] http://www.diagnosticstrategies.com/info_retrieval.htm

[Fagan, 1987] Fagan, Joel L. "*Automatic Phrase Indexing for Document Retrieval: A Examination of Syntactic and Non-Syntactic Methods*". PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York 14853-7501, 1987.

[Guarino et al., 1999] Guarino, N., Masolo, C., Vetere, G., "*Content-Based Access to the Web*", IEEE Intelligent Systems, May/June 1999, pp.70-80.

[Hiemstra & Vries, 2000] Hiemstra, D. and Vries, A. P., "*Relating the new language models information retrieval to the traditional retrieval models*" published as CTIT technical report TR-CTIT-00-09, May 2000, available http://www.ctit.utwente.nl

[Krovetz, 1992] Krovetz, R. "*Lexical Ambiguity and Information Retrieval*" ACM Transactions on Information Systems, 10(2): 115-141, 1992.

[Lee, 1994] Lee, J.H., "*Properties of Extended Boolean Models in Information Retrieval*", ACM Conference on Research and development in information retrieval, ACM-SIGIR, pp.182-190, 1994.

[Lewis, 1996] Lewis, D., "*Natural Language Processing for Information Retrieval*" Communications of the ACM, 39(1):92-101, Jan 1996.

[Lucene]:

[1] http://jakarta.apache.org/lucene/docs/index.html

[2] http://www.darksleep.com/puff/lucene/lucene.html

[3] http://lucene.sourceforge.net/cgi-bin/faq/faqmanager.cgi

[4] http://haystack.lcs.mit.edu/papers/Adar.thesis/node14.html

[5] http://www.onjava.com/pub/a/onjava/2003/03/05/lucene.html

[6]http://www.arielpartners.com/arielpartners/content/public/topics/technology/technologyReviews/lucene

[Mitra et al., 1997] Mitra, M., Buckly, C., Singhal, A., Cardie, C., "*An Analysis of Statistical and Syntactic Phrases*", National Science Foundation under grant IRI-9624639, 1997

[Macklovitch, 2001] Elliott Macklovitch, "*The New Paradigm in NLP and its Impact on Translation Automation*", 2001, http://rali.iro.umontreal.ca/Publications/urls/macklovi-symp-proceed.doc

[Nie, 2001] Nie, J.Y., "*A general Logical approach to inferential Information Retrieval*", Encyclopedia of Computer Science and Technology, eds. Kent E. A. and Williams J.G., Vol. 44, 2001, pp:203–226. http://www.iro.umontreal.ca/ ~nie/publication.html

[Nie, 2002] Nie, J. Y., Dufort, Jean-François, "*Combining Words and Compound Terms for Monolingual and Cross-Language Information Retrieval*", Information *2002*, 2002, http://www.iro.umontreal.ca/~nie/publication.html

[Nstein 2003] "Concepts Extractor" documentation, Nstein technology Inc., 2003

[Ozaku et al., 2003]Hiromi itoh Ozaku, Masao Utiyama, Hitoshi Isahara "*Study on Merging Multiple Results from Information Retrieval System*" Proceedings of the Third NTCIR Workshop (cc:2003 National Institute of Informatics).

[Ozaku et al., 03] Ozaku, Hiromi iroh, Utiyama, M., Isahara, H., Kono,Y., Kidode, M., "*A Comparative Study on Merging Results from WWW Retrieval System*" Communications Research Laboratory, Nara Insititute of Science and Technology, 2003

[Pickens & Croft, 2000] Pickens, J. and Croft, B. "*An Exploratory Analysis of Phrases in Text Retrieval*" http://citeseer.ist.psu.edu/pickens00exploratory.html November 5, 2000

[Paice, 1984] Paice, C.P., "*Soft evaluation of Boolean search queries in information retrieval system*". Information Technology: Research and Development, 3(1):33-42, 1984.

[Porter, 1980] An algorithm for suffix stripping, Program, 14(3): 130-137, 1980. http://www.muscat.com/~martin/stem.html

[Strzalkowski, 1995] Strzalkowski, T., "*Natrual Language Information Retrieval*", Information Processing & Management, 31(3): 397-418, 1995.

[Salton & McGill, 1983] Salton, G. and McGill, M.J., "Introduction to Modern Information Retrieval". Mcgraw-hill`Book Company, New York, 1983.

[Singhal et al., 1995] Amit Singhal, Chris Buckley, Mandar Mitra " *Pivoted Document Length Normalization*", Technical Report: TR95-1560, 1995. http://ir.iit.edu/~dagr/cs529/files/handouts/singhal96pivoted.pdf

[Salton et al., 1990] Salton, G., Buckly, C., and Smith, M, "*On the application of syntactic methodologies in automatic text analysis*" Information Ptocessing & Management, 26(1):73-92, 1990.

[Salton et al., 1975] Salton, G., Wong, A. and Yang, C.S., "*A vector space model for information retrieval*". Journal of the American society for Information Science. 18(11): 613-620, 1975.

[Strzalkowski et al., 2000] Tomek Strzakowski, Jose Perez-carballo, Jussi Karlgren "*Natural Lanhuage Information Retrieval: TREC-8 Report*", 2000.

[Salton & Buckley, 1988] Salton, G. & C. Buckley "*Term-weighting approaches in automatic text retrieval*", Information Processing and Management 24, 513–523,1988.

[Si & Callan, 2003] SI, LUO and CALLAN, J., " *A Semisupervised Learning Method to Merge Search Engine Results*", ACM Transactions on Information Systems, 21(4) : 457-491, October 2003

[Si & Callan, 2002] SI, LUO and CALLAN, J., " *Using Sampled Data and Regression to Merge Search Engine Results*", SIGIR 02, August 11-15, 2002

[Strzalkowski et al., 1996] Strzalkowski1, T., Guthrie, L., Karlgren, J., Leistensnider, J., Lin, F., Perez-Carballo, J., Troy Straszheim, T., Wang, J., Wilding, J., "*Natural Language Information Retrieval: TREC-5 Report*", 1996.

[Strzalkowski et al., 1998] Tomek Strzalkowski, Qees Stein, G. Bowden Wise "*Natural Language Information Retrieval: TREC-7 Report*", 1998

[Su, 2002] Su, D. Chi-Chuan, "*Performance Analysis and Optimization on Lucene*" JakartaLucene@1999-2002. http://www.stanford.edu/class/archive/cs/cs276a/cs276a.1032/projects/reports/dsu800.pdf

[Savoy et al., 1997] Savoy, J., Calve, A. Le, Vrajitoru, D., "Report on the TREC-5 Experiment: Data Fusion and Collection Fusion", Proceedings of the TREC-5, D. Harman (ED) NIST Publication500-238, Gaithersburg (MD), pp. 489-502, 1997

[Woods, 1997] Woods, W. A. "*Conceptual Indexing: a better way to organize knowledge*," technical Report SMLI TR-97-61, Sun Microsystems Laboratories, Mountain view, CA, April 1997. (Available online at: http://www.sun.com/research/techrep/1997/abstract-61.html).

[Yu & Salton, 1977] Yu, C.T., Salton, G., "*Effective Information Retrieval Using Term Accuracy*" Communications of the ACM, 20(3):135-142, 1977.

[Zhai et al., 1997] Zhai, C. X., Tong, X., Milic-Frayling, N., Evans, D.A., "*Evaluation of Syntactic Phrase Indexing*", CLARIT NLP Track Report, 1997